UNIVERSITAT POLITÈCNICA DE CATALUNYA

DOCTORAL THESIS

A Deep Dive into Ethereum's PoS Transition: Protocol Design Choices and Their Empirical Unexpected Limitations

---

Author:
Mikel CORTES-GOICOECHEA

Supervisor:
Dr. Jose Luis MUÑOZ TAPIA
Dr. Leonardo BAUTISTA-GOMEZ

A thesis submitted in fulfillment of the requirements for the degree of Network Engineering

in the

Information Security Group
Network Engineering Department

July 23, 2024

Declaration of Authorship

I, Mikel CORTES-GOICOECHEA, declare that this thesis titled, "A Deep Dive into Ethereum's PoS Transition: Protocol Design Choices and Their Empirical Unexpected Limitations" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: July 23, 2024

"Why don't blockchain enthusiasts throw parties? Because you can't centralize fun in a peer-to-peer network!"

ChatGPT

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

Barcelona School of Telecommunications Engineering
Network Engineering Department

Network Engineering

A Deep Dive into Ethereum's PoS Transition: Protocol Design Choices and Their Empirical Unexpected Limitations

by Mikel CORTES-GOICOECHEA

The advent of the internet, marked by pivotal developments such as the launch of Arpanet and the standardization of HTTP, has irrevocably changed the fabric of modern society. Centralized platforms like Microsoft, Google, Apple, and Amazon have dominated this digital landscape, offering many services ranging from cloud computing to online storage. However, the centralized nature of these services has raised significant concerns regarding user privacy, data integrity, and the potential for censorship. In response to these issues, the open-source community has explored peer-to-peer alternatives, notably in the realm of distributed file systems, ledgers, and blockchain technology. Blockchains, popularized by the emergence of Bitcoin, promote a democratized service model that challenges the centralized status quo. Yet, they are not without their own challenges, including decentralization, security, privacy, and performance.

This thesis delves into the nuances of blockchain technology, focusing on Ethereum's transition from Proof of Work (PoW) to Proof of Stake (PoS) and its implications on network hardware requirements, topology, and overall performance. The development of Ethereum serves as a small-scale reflection of the broader ambitions and challenges in transitioning to Decentralized Finance (DeFi) platforms. Despite significant theoretical advancements in consensus mechanisms and scalability solutions, real-world implementations and experimental validations remain sparse. This work aims to bridge this gap by comprehensively analysing Ethereum's PoS transition by examining the interlaced relationships between software logic, hardware configurations, and network dynamics. Through novel measurement models and tools, this thesis contributes to a deeper understanding of how Ethereum's architectural changes impact its ecosystem and its participants' behaviours.

Lastly, the research presented in this thesis illustrates the technical and operational challenges facing Ethereum and similar blockchain platforms and proposes a series of contributions that advance the field. This work empirically analyses the future enhancements in blockchain technology by exploring the implications of the network and its topology, to the viability of decentralized validation processes, and the potential for scaling solutions like Data Availability Sampling. The open-source tools and methodologies developed within the thesis scope represent the commitment to transparency and collaboration, which follows the spirit of the decentralized communities it seeks to serve. Through a mix of theoretical exploration and empirical research, this thesis aims to provide a deeper and more detailed understanding of Ethereum PoS' design choices, its capabilities and the limitations this one represents in future steps and upgrades, leading the way for more resilient, scalable, and decentralized digital infrastructures. ...

Acknowledgements

My heartfelt thanks to my thesis supervisors, Leonardo Bautista and Jose L. Muñoz, for their guidance, support, comments, and dedication and for opening my mind and the doors to such an incredible community and research area. They didn't just trust my abilities but also gave me an unbelievable opportunity to show and discover what I was capable of.

I'm incredibly grateful to the BSC Computer Architecture and Resilience in Distributed Systems Department members for hosting me over these last four years of my life. Particularly Dr Adrian and Dr Osman for handling all my annoying emails and helping me troubleshoot all the HR tickets. Also, to the colleagues I encountered over these years: Sarkawt, Luca, Albert, Kai, Elias, Julian, Jonathan, Noe, Xabi, Luis, and Andres. Their camaraderie and engaging discussions during our shared meals and gatherings greatly enhanced my research journey.

To the MigaLabs team Tarun Mohandas, Josep Chetrit, Iuri Pons, Marina Terentii, Santiago Somoza, and Yulliana for the support and the colossal motivation they transmitted to research on the Ethereum's ecosystem, leading to many cool projects, collaborations and papers.

To Protocol Labs and the ProbeLab team, Yiannis Psaras, Dennis Trautwein, Ian Davis, and Guillaume Michel, for giving me the incredible opportunity to research p2p networks surrounded by the leading heads of such relevant projects such as Libp2p and IPFS, and for introducing me that warmly to the entire community.

To the Status.im and the Codex-Storage team, especially to Csaba Kiraly and Dmitriy Ryajov, for the fascinating opportunity to do a research internship on edge-leading Ethereum's scalability solutions.

To Cambridge University and especially to Alexander Neumüller, who trusted our small research group to develop one of the coolest studies on the energy consumption and the carbon footprint of Ethereum and who also supported the development of some of the core components of this thesis and gave me the chance to make an internship at his research group.

To the Ethereum core developers and the community who helped the entire MigaLabs team and me shape and ship all our research, providing vital guidelines and feedback on our studies. In particular, I'd like to thank Protolambda for dedicating time to a young master's student who was learning and Paritosh for his unconditional support to the entire team.

I wouldn't like to forget about other supporting teams and companies like Attestant, Lido, Obol, and Metrika, who believed in the team and our research direction and proactively supported us in a wide variety of ways.

Finally, I'd like to thank my family and close friends, who, despite thinking I was playing around with cryptocurrencies and losing money on scams, supported me and were present to celebrate the good moments and to cheer me up in the bad ones.

Contents

Declaration of Authorship iii

Abstract viii

Acknowledgements ix

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Challenges . . . 4
- 1.3 Objectives . . . 5
- 1.4 Contributions . . . 5

2 Background 9
- 2.1 Peer-to-Peer Networks . . . 9
  - 2.1.1 Journey of Distributed Networks . . . 9
  - 2.1.2 Definition and Characteristics of P2P Networks . . . 10
  - 2.1.3 Challenges to Overcome . . . 10
- 2.2 Blockchain . . . 11
  - 2.2.1 Prime on Blockchain Technology . . . 11
  - 2.2.2 Reaching Consensus . . . 12
- 2.3 Ethereum . . . 13
  - 2.3.1 Ethereum's Transition to PoS . . . 14
  - 2.3.2 Protocol Defined Slot Time Ranges . . . 17
  - 2.3.3 Ethereum validators and testing networks . . . 18
- 2.4 Networking protocols . . . 19
  - 2.4.1 Libp2p . . . 19
  - 2.4.2 GossipSub . . . 20
  - 2.4.3 Ethereum Node Records (ENR) . . . 21
  - 2.4.4 DHTs . . . 21
    - Kademlia DHT . . . 22
  - 2.4.5 Discovery5 . . . 22
  - 2.4.6 The InterPlanetary File System (IPFS) . . . 23

3 Fundamentals of Ethereum's P2P Network 25
- 3.1 Introduction . . . 25
- 3.2 Background . . . 26
- 3.3 Methodology . . . 27
  - 3.3.1 Network Crawler . . . 27
- 3.4 Evaluation . . . 29
  - 3.4.1 Ethereum's Public DHT . . . 29
  - 3.4.2 Network Metrics . . . 32
  - 3.4.3 Diversity . . . 34
- 3.5 Conclusion And Future Work . . . 36

4 Evolution of Hardware Requirements with Ethereum's PoS Transition 37
4.1 Introduction . . . 37
4.2 Related Work . . . 38
4.3 Methodology . . . 39
4.3.1 Different Users, Different Hardware . . . 40
4.3.2 Client Configurations . . . 41
4.3.3 Experiments Timing Setups . . . 41
4.3.4 Tooling . . . 43
Node Exporter . . . 43
API Workloader . . . 43
4.4 Hardware Resources' Evaluation . . . 44
4.4.1 Synchronization process . . . 45
CPU Utilization . . . 45
Memory Utilization . . . 46
Disk utilization . . . 48
Network Bandwidth . . . 50
Performance . . . 50
4.4.2 The Bigger, the Better? . . . 51
Sync Speed . . . 51
Disk Utilization . . . 52
4.4.3 How Small Could We Go? . . . 54
Beacon Chain Synchronization on a Raspberry Pi 4 . . . 55
CPU Utilization . . . 56
Memory Utilization . . . 56
Performance . . . 57
4.4.4 Regular performance . . . 57
CPU Utilization . . . 59
Memory Utilization . . . 59
Disk Utilization . . . 62
Network Bandwidth . . . 62
4.5 Hardware and network observations . . . 64
4.5.1 CPU at Slot Time Utilization . . . 64
4.5.2 Hardware Resources Evolution over Hardforks . . . 66
4.5.3 Perceiving Network Instabilities from Chain Synchronization . . . 67
4.5.4 Beacon API Performance . . . 72
4.6 Discussion . . . 73
4.6.1 Strengths and Weaknesses of the Clients . . . 74
4.6.2 Hardware Recommendations . . . 76
4.6.3 The Network can Benefit from your Client Choices . . . 76
4.7 Conclusion . . . 77

5 Incentive Changes with Ethereum's Merge 79
5.1 Introduction . . . 79
5.2 Related Work . . . 80
5.3 Prime on Ethereum's Rewards . . . 81
5.3.1 Execution Layer's Block Proposal Rewards . . . 81
5.3.2 Characterization of Attestation Rewards . . . 82
5.3.3 Characterization of Sync Committees Rewards . . . 83
5.3.4 Characterization of Block Proposals . . . 84
5.3.5 Validators' Penalizations . . . 85
5.3.6 Maximum Extractable Reward (MER) . . . 86

5.3.7 GotEth: Ethereum Chain Indexer . . . 86
5.3.8 Validator to Staking Pools Mapping . . . 88
5.4 Performance of Validators during “The Merge" . . . 90
5.4.1 Descentralization of the Consensus . . . 90
5.4.2 Validators Overall Performance . . . 91
Missed Attestation Flags . . . 92
MER . . . 92
Missed Block Proposals . . . 92
5.4.3 Empirical Distribution of the Rewards in a post-Merge Scenario 93
5.4.4 Concatenation of Multiple Block Proposals . . . 93
5.4.5 MER per Staking Entities . . . 95
5.5 Conclusion and Future Work . . . 95

6 Impact of the P2P Network on the Ethereum’s Decentralization 97
6.1 Unveiling Ethereum’s Hidden Centralization Incentives: Does Connectivity Impact Performance? . . . 97
6.1.1 Introduction . . . 97
6.1.2 Related Work . . . 99
6.1.3 Methodology . . . 100
Scoring Ethereum CL’s Duties . . . 100
Deployment of the nodes . . . 102
6.1.4 Validator Consensus Performance . . . 103
Reward Comparison Based on Goerli Validators . . . 103
6.1.5 Block Scores . . . 106
Beacon Block Generation Differences . . . 107
6.1.6 Conclusion . . . 111
6.2 DVT in Ethereum’s PoS: Gains and Loses . . . 112
6.2.1 introduction . . . 112
6.2.2 Distributed Validator Technology: How Does It Work? . . . 113
6.2.3 Methodology . . . 115
Software services . . . 115
6.2.4 Performance Indexes . . . 116
Data Collection Tools . . . 117
6.2.5 DVT Cluster VS Canonical Validator Setup Comparison . . . 119
Steadyness of the Validator . . . 119
DVT Isolated Performance . . . 125
6.2.6 Nodes’ Performance . . . 127
Limit of DVT Validators per Cluster . . . 127
6.2.7 Conclusion . . . 129
6.3 Conclusion . . . 130

7 Exploring the Future of Ethereum’s Scalability 131
7.1 Introduction . . . 131
7.2 How does a DHT fit in Ethereum’s DAS approach? . . . 132
7.3 Methodology . . . 133
7.3.1 DHT Simulator . . . 133
7.3.2 Verification of the DHT Simulations . . . 135
7.3.3 Development of the experiments . . . 136
7.4 Evaluation . . . 136
7.4.1 DHT lookups . . . 136
7.4.2 Seeding of the DHT . . . 140

7.4.3 DHT limitations . . . 142
Single seeder to the DHT . . . 142
Multiple seeders to the DHT . . . 145
Avoiding the DHT . . . 145
7.5 Conclusion . . . 145

8 Conclusions **147**
8.1 Summary of Achievements . . . 147
8.2 Notable Observations . . . 149
8.3 Future Outlooks . . . 149

A Publications and Dissemination **151**
A.1 Publications . . . 151
A.2 Workshops and Talks . . . 152
A.3 Research Collaboration . . . 152
A.4 Grants . . . 152

B Provider Record's Liveness in IPFS **153**

Bibliography **155**

List of Figures

1.1 Most remarkable hits in the history of the Internet. 1
1.2 Evolution of users of the internet over time. 3

2.1 Block concatenation in blockchain technology. 13
2.2 Ethereum transition from PoW to PoS. 14
2.3 Subdivision in layers of Ethereum after the merge. 15
2.4 Assignation of active validators into attestation committees. 16
2.5 Expected temporal division of a slot, including the time each duty should be sent and propagated over the network. 18
2.6 Ethereum network's hard-forks organization, public testnets in red, Goerli testnet at the bottom. Note that hardforks are tested on testnets before they occur on mainnet. 19

3.1 Diagram of the Armiarma crawler's internal structure and its interaction with the network. 27
3.2 Number of ENRs actively shared with the Crawler in the last 24 hours. Snapshot from the 16th of February 2024. 30
3.3 Number of ENRs that are not deprecated. Nodes with ENRs were actively shared on the discv5 protocols, plus the nodes we could connect with within the last three hours. 30
3.4 Number of active nodes in the Ethereum Network. Nodes that weren't deprecated at the snapshot time and that were successfully connected and identified at least once. 31
3.5 Error distribution of the individual connection attempts to non-deprecated peers. Snapshot from the 16th of February 2024. 32
3.6 Geographical distribution of the Ethereum active nodes. Snapshot from the 16th of February 2024. 32
3.7 Distribution of Round Trip Time (RTT) between the active nodes and the crawler. Snapshot from the 16th of February 2024. 33
3.8 Distribution of IPs shared across the active nodes in the network. Snapshot from the 16th of February 2024. 34
3.9 Distribution of the advertised number of attestation subnets' subscriptions from active nodes in the network. Snapshot from the 16th of February 2024. 35
3.10 Distribution of different ISPs for the active nodes in the network. Snapshot from the 16th of February 2024. 35
3.11 Distribution of client software used by active nodes in the network over time. 36

4.1 CPU utilization while syncing the chain from a default node. Comparison across clients. 44
4.2 Memory utilization while syncing the chain from a default node. Comparison across clients. 45

4.3 Disk read operations/s while syncing the chain from a default node. . 46
4.4 Disk write operations/s while syncing the chain from a default node. . 47
4.5 Network inwards utilization while syncing the chain from a default node. 48
4.6 Network outwards utilization while syncing the chain from a default node. 49
4.7 Chain synchronization speed as the number of slots downloaded and processed since the node on standard hardware was run. 52
4.8 Chain synchronization speed as number of slots downloaded and processed since the fat node was run. 53
4.9 Slot synchronization ratio per second for the different clients. 53
4.10 Disk read operations by the clients while running in a “Fat" node. . . . 54
4.11 Disk write operations by the clients while running in a “Fat" node. . . 55
4.12 CPU utilization while syncing the chain in a Raspberry Pi 4. 56
4.13 Memory utilization while syncing the chain in a Raspberry Pi 4. 57
4.14 Disk writing speeds while syncing the chain in a Raspberry Pi 4. 58
4.15 Chain synchronization speed as the number of slots downloaded and processed since the node on the Raspberry Pi 4 was run. 58
4.16 CPU usage by clients following the head of the chain. 59
4.17 Memory usage by clients following the head of the chain. 60
4.18 Disk reads by clients following the head of the chain. 60
4.19 Disk writes by clients following the head of the chain. 61
4.20 Concurrent node connections by the clients while following the head of the chain. 61
4.21 Network incoming bandwidth for the clients while following the head of the chain. 63
4.22 Network incoming bandwidth for the clients while following the head of the chain. 63
4.23 CPU utilization at the slot range while following the head of the chain. 65
4.24 Chain synchronization time by clients in the Medalla testnet. 68
4.25 Zoomed slot synchronization speed of consensus clients in the Medalla testnet. 68
4.26 Disk usage of consensus clients during the Medalla testnet synchronization. 69
4.27 Disk Usage of Ethereum clients while syncing the Medalla testnet using the slots as X axis. 70
4.28 Number of times a block is queued to be persisted while syncing the Medalla chain on Lighthouse. Zoomed at the non-finality period. . . . 71
4.29 Events timeline for Lighthouse while syncing the Medalla testnet. Zoomed at the non-finality period. 71
4.30 Distribution of the Beacon node’s response time under a single concurrent request workload. Note that 20 seconds was the timeout of the request. 73
4.31 Distribution of the Beacon node’s response time under ten concurrent requests workload. Note that 20 seconds was the timeout of the request. 74

5.1 Evolution of rewards for different layers in Ethereum. 81
5.2 Rewards extracting schema followed by the state analyzer tool for Epoch $n-1$ (in red: actions of our indexer tool; in blue, actions at the protocol). 82

5.3 Internal diagram of the Ethereum chain indexer tool. In green, the internal structure of the tool showcasing the five main steps: 1) identify when each epoch and slot starts, 2) download of the epoch duties, the beacon block on each slot (if it exists), the state at the end of the epoch, and the execution receipts, 3) with the raw chain data, compose the participation of each validator and the reward that each one got, 4) based on the chain state, compose the MER of each validator, and 5) persist of the metrics into the database. 87
5.4 Block proposal's distribution per mining and staking entities for the two months of study. 88
5.5 Distribution of active validators across the known staking entities at 16th of February 2024 89
5.6 Time distribution of missed flags percentage from the total number of validators. Aggregating the total missed flags on the top chart and distinct missed flags at the bottom. 90
5.7 Number of missed blocks pre- and post-merge. 91
5.8 Maximum Extractable Reward for validator network over the $14k$ epochs. 91
5.9 Empirical distribution of rewards from different sources. 93
5.10 Maximum Extractable Reward for the eight biggest staking entities. . . 94

6.1 Internal diagram of the Ethereum's block scorer tool. In green, the internal structure of the tool showcasing the five main steps: 1) identify when each epoch and slot starts, 2) download the epoch duties, the beacon block on each slot (if it exists), the state at the end of the epoch, and the execution receipts, 3) with the raw chain data, compose the participation of each validator and the reward that each one got, 4) based on the chain state, compose the MER of each validator, and 5) persist of the metrics into the database. 101
6.2 Average missed attestation flags. 104
6.3 Average achieved reward per location. 105
6.4 Aggregated missed blocks per location. 106
6.5 Number of chain reorganizations per location. 107
6.6 Average block score per location. 108
6.7 Average arrival latency of block messages inside slot time per location. 108
6.8 Number of new votes and their correct flags on each location. 109
6.9 Percentage of slots each beacon node was out of sync in each location. 110
6.10 Average block arrival time per client and location. 111
6.11 Internal communication between the beacon node, Charon, and the validator client when performing a consensus duty of a DVT validator. 114
6.12 Percentage of missed attestation flags for the first part of the experiment with Charon on its version v0.14.0. The upper figure shows the missed flags across the different validator sets. In the bottom one, the missed flags are aggregated by types of clusters (non-DVT vs DVT/Obol). 121
6.13 Percentage of missed attestation flags for the first part of the experiment with Charon on its version v0.15.0. The upper figure shows the missed flags across the different validator sets. In the bottom one, the missed flags are aggregated by types of clusters (non-DVT vs DVT/Obol). 122
6.14 Cumulative distribution function of the attestation inclusion delay per DVT cluster, with both versions aggregated. 122

6.15 Percentage of successful block proposals of Charon on its version $v0.14.0$. On the left, the successful block proposals are aggregated by cluster. On the right, they are aggregated by validating technology. 123
6.16 Percentage of successful block proposals of Charon on its version $v0.15.0$. On the left, the successful block proposals are aggregated by cluster. On the right, they are aggregated by validating technology. 124
6.17 Ratio of achieved rewards with the MER of the validators aggregated by cluster on the left and by validator technology on the right on Charon's $v0.14.0$ version. 125
6.18 Ratio of achieved rewards with the MER of the validators aggregated by cluster on the left and by validator technology on the right on Charon's $v0.15.0$ version. 126
6.19 Latency heatmap across the different cluster nodes using Charon on its version 0.14.0. On the left, aggregated by regions: APAC (Asia–Pacific), EMEA (Europe, the Middle East and Africa) and NAFTA (North American Free Trade Agreement). On the right, aggregated by city. 126
6.20 Latency heatmap across the different cluster nodes using Charon on its version 0.15.0. On the left, aggregated by regions: APAC (Asia–Pacific), EMEA (Europe, the Middle East and Africa) and NAFTA (North American Free Trade Agreement). On the right, aggregated by city. 127
6.21 Beacon score of each Charon client participating in the four-node cluster per step in the experiment. 128

7.1 Block split proposal for DankSharding's DAS. 134
7.2 Description of the DHT components in the *dht* simulator, and an example of the internal logic of their interaction. 134
7.3 CDF of the DHT hops while performing 100 concurrent retrievals in the IPFS network. 137
7.4 CDF of the time to finish 100 sets of 80 concurrent retrievals from the IPFS network. 137
7.5 CDF of the simulated number of hops while looking for 200 concurrent keys with different overheads with $k = 20$, $alpha = 3$, and $beta = 20$. 138
7.6 Accuracy of the DHT simulator model matching lookup performances in the IPFS network from different locations while keeping $k = 20$, $alpha = 3$, and $beta = 20$. Note that "fer" stands for "fast error rate". . . 139
7.7 CDF of the simulated 100 DHT retrieval sets with 200 concurrent keys each using different overheads. The DHT simulation had the following parameters: $k = 20$, $alpha = 3$, and $beta = 20$. 140
7.8 CDF of the simulated 1.000 concurrent DHT provide operations from a single node. 141
7.9 CDF of the simulated 10.000 concurrent DHT provide operations from a single node. 141
7.10 CDF of the simulated 262.144 concurrent DHT provide operations from a single node. 143
7.11 CDF of the time to finish 100 sets of 80 concurrent CID provides to the IPFS network. 143

List of Tables

4.1 Hardware configuration of the control machines. 40
4.2 Running dates of each client during the synchronization of the Medalla network. 43
4.3 Total disk usage of the beacon node's database keeping beacon blocks and states after syncing the chain on the default mode. 49
4.4 Target of peer connections for each client during the chain synchronization on a default configuration. 49
4.5 Target of peer connections for each client during its regular operation. 63
4.6 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. 66
4.7 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. 67
4.8 Percentage of responses the Beacon node could successfully reply under different concurrent request workloads. 72

5.1 Table with the number of validators proposing blocks consecutively. (Pre-merge and Post-merge comparison) 94
5.2 Table with the number of consecutive blocks proposed by the larger staking entities. 95

6.1 Table with the versions used during the studies. 102
6.2 Hardware setup per location for the rewards study. 103
6.3 Hardware type on each tested location of the block score study. 103
6.4 Hardware specifications of different cloud service providers. 117
6.5 Cloud provider hosting the four-node DVT cluster. 118
6.6 Cloud provider hosting the seven-node DVT cluster. 118
6.7 Cloud provider hosting the ten-node DVT cluster. 118
6.8 Table summarizing all the exact block proposal duties per validator cluster, including the number of missed blocks over the whole study. . 124
6.9 Rewards during scaling experiment with and without filtering block proposals and sync committee rewards. 129
6.10 Number of missed attestation per different validator set. 129

List of Abbreviations

EF	**E**thereum **F**oundation
EIP	**E**thereum **I**mprovement **P**roposal
ENR	**E**thereum **N**ode **R**ecords
DAS	**D**ata **A**vailability **S**ampling
DApp	**D**ecentralized **App**lication
DeFi	**De**centralized **Fi**nance
DHT	**D**istributed **H**ash **T**able
DVT	**D**istributed **V**alidator **T**echnology
IPFS	**I**nter **P**lanetary **F**ile **S**ystem
P2P	**P**eer-to-**P**eer
PoW	**P**roof of **W**ork
PoS	**P**roof of **S**take
TTL	**T**ime **T**o **L**ive
WWW	**W**orld **W**ide **W**eb

To my mum and dad, for sacrificing everything and unconditionally supporting me in whatever made me happy. And to any young person who has a dream and thinks it isn't possible, only hard work, dedication, and time will make it happen.

Chapter 1

Introduction

The internet has considerably evolved over the past six decades since the first apparition of Arpanet [200] and the standardization of the *HTTP* protocol with the publication of the first website, as shown in figure 1.1. In a centralized landscape where business models and e-commerce platforms [159] have consolidated as a relevant sector in today's economy, not everything that comes with the accessibility and low costs of services provided by large centralized entities such as Amazon, Apple, Google, or Microsoft, is as shiny as it looks. The mass adoption of such services, such as cloud computing, storage, online shopping, and even human interaction, has not only changed our day-to-day lives making it easier. But it has also negatively affected our freedom to many degrees. These companies collect enormous amounts of information from all these user interactions, jeopardizing the identity and privacy integrity of the users [54]. Reaching, as some past experiences have shown [81], to alter or manipulate our behaviours or interactions with our equals. Moreover, the centralization degree that our society is reaching, where more and more citizens rely on these entities for daily or business tasks, can be turned around and positioned against these same users if access to these services is revoked. Let's remember that these private companies legally have the right to stop providing service to any user not following the so popular *Terms of Service*. This, ultimately, could lead to some censorship or ban of any user or content from their platforms.

Under the premise of democratizing such relevant services that these centralized entities offer, the open-sourced community has put a lot of effort over the last two decades into proposing and improving a range of peer-to-peer alternatives. Many use cases and examples exist for these distributed or decentralized applications. The first and most popular are distributed storage networks, where users without

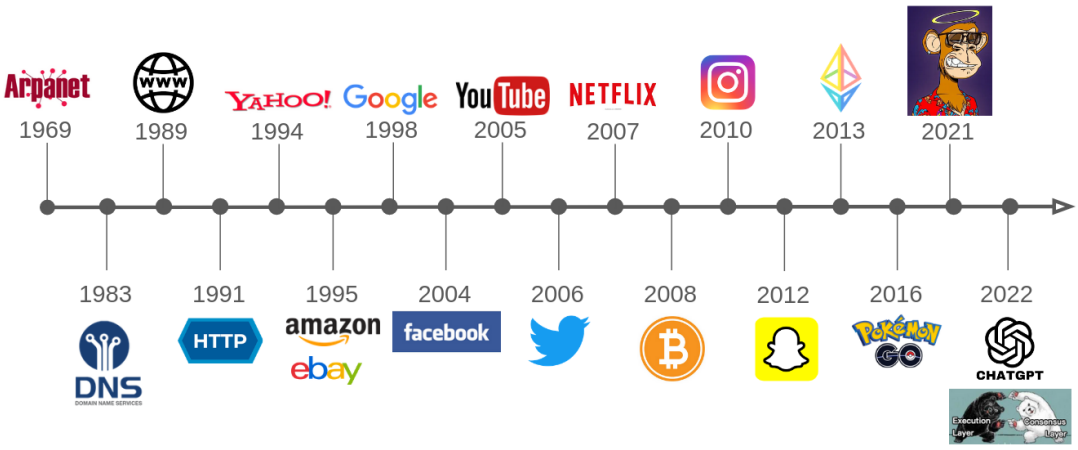

FIGURE 1.1: Most remarkable hits in the history of the Internet.

any central organization share and download content files using direct connections. In the scope of this thesis, we will dig into the recently popularized different approaches, distributed ledgers or blockchains.

Blockchains, heavily popularized by Satoshi Nakamoto and the first implementation of Bitcoin [131], have precisely tried to achieve that democratization of services. After the big economic crash of 2008, Bitcoin was presented as a permissionless and decentralized alternative that could prevent, to some degree, the lack of judgment that central banks offered. As community-driven platforms, blockchains generally require a high degree of decision acceptance (we could see a 51% acceptance for Bitcoin or a 67% for Ethereum) to apply any measures or changes in the protocol.

However, as with most of the peer-to-peer networks, blockchains also have big drawbacks: distributing these services across multiple users who now become service providers isn't an easy task. In fact, there are significant compromises that generally need to be made.

The first one is related to the **decentralization** of the services. Building a decentralized and distributed alternative finance platform undoubtedly attracts investors. This undoubtedly helps translate the wealth distribution of the canonical economy to the alternative one. Despite early adopter individuals, it is a matter of time before large investors arrive or will arrive in the ecosystem, becoming a significant share of the service providers. This isn't strictly bad, as the network and platform are common and shared across multiple large actors. However, it could certainly become a problem if the targeted decentralization is not accomplished.

The second relates to the **security and privacy** aspects of participating in public blockchains. Despite the identity of users being "hidden" under public keys, this "pseudo-anonymity" is, to some point, exposed as these transactions can expose the IP from which it was sent. Furthermore, the transparency and immutability promoted by these ledgers might expose data from non-careful users. On the security aspect, the democratic factor of these networks can potentially turn around and go against the network itself if an entity or a malicious user reaches control over the majority of the network. Furthermore, the human factor in the automation of critical code and the lack of existing regulations that these networks currently have led to possible code vulnerabilities or the exposition of users to social engineering attacks.

The third most common one is related to the **performance** of these networks. They are generally known for their scalability issues while maintaining security and decentralization. This known trilemma often limits the throughput these networks can offer the end users as they try to prioritize the other two factors. Since the network's security highly relies on the redundancy factor of the network, i.e., all the validators need to compute and validate individual events, and all of them keep a local copy of the chain, these platforms are highly inefficient. Especially on the first iterations, such as Bitcoin and its Proof of Work consensus mechanism, well known for its heavy energy hungry network and its sustainability issues.

1.1 Motivation

The apparition of the internet and the mass adoption of this one has become one of the most relevant changes in human history [108, 72]. Humans have always had a collaborative nature [86, 24, 191], since the early beginnings of our existence, we've united efforts to ensure not only the survival of the kind but to improve our living conditions [192]. This cooperation, over centuries, has led us to the current modern

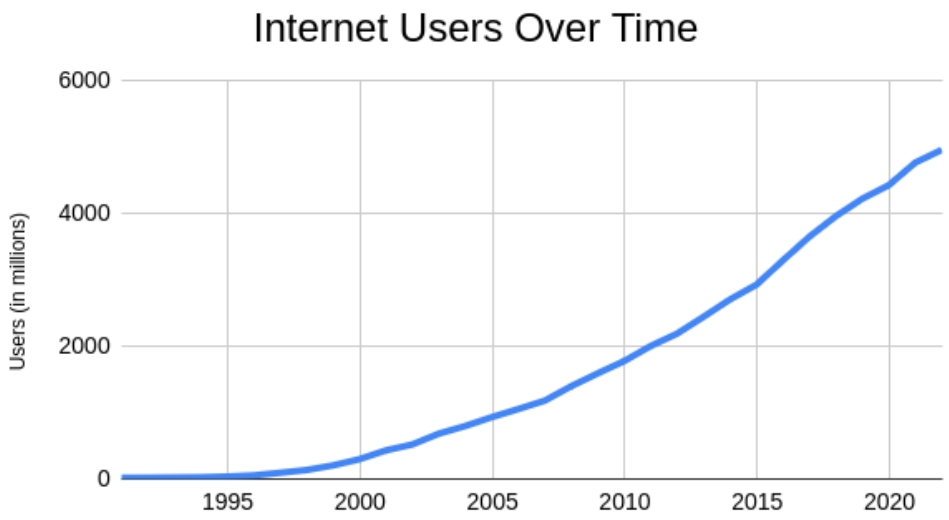

FIGURE 1.2: Evolution of users of the internet over time.

era, where, with the help of the internet, not only knowledge but freedom of speech and multiple tools have been granted to the biggest part of the world population [42, 161].

This significant technological advance, together with the mass adoption of it through the world population shown in figure 1.2, led to globalization and the compilation of a whole new modern era for communication between humans, the world's economy, and the fastest growth in technological advances in our history. Blockchains have come to stay in such a preexisting atmosphere, promoting an open-sourced and decentralized alternative to the current economic system, which precisely relies on the collaborative nature of humans.

Although blockchain faces all the previously introduced three main challenges, there have been massive research improvements over these last 12 to 15 years. Most of those improvements were made with the expected final goal of making blockchain a real competitor to the canonical financial system. Among these improvements, we could highlight the heavy theoretical research on the consensus mechanisms, which haven't only made it possible to separate from the heavy carbon footprint of PoW. Some chains, including Ethereum, have significantly contributed to achieving more sophisticated while optimising consensus algorithms over distributed validators without compromising the public ledger's decentralisation and security.

Despite having multiple viable solutions, at least on paper, there aren't many real implementations that have gone beyond the theory and the simulations. Ethereum stands as a leading platform and community in that regard, not only proposing a novel Proof of Stake (PoS) theoretical approach but also consolidating the idea with many major tweaks to make a whole PoW to PoS transition viable. This transition didn't only imply a vast reduction in the overall energy consumption of the network by 99%, but it expanded the horizons of scalability solutions.

Regardless of all the improvements gathered in Ethereum's extensive roadmap, the PoW to PoS transition still means a radical change in the network that has not yet been fully understood. Theory and simulations are hard to fully extrapolate to real implementations, especially on hard-to-measure networks like peer-to-peer ones. Thus, there is still a huge lack of experimental work on understanding, profiling, and modelling all the changes and limitations these new blockchain solutions

offer.

This thesis focuses on addressing those particular experimental gaps in Ethereum's narrative. The thesis presents novel measurement models to better understand how this transition to a more sustainable platform has impacted the network hardware requirements of participating in the network and the impact that this one has on the network's topology. Furthermore, it explores how the existing networking protocols and solutions react to these more complex consensus solutions and the bottleneck or limiting factor for the performance of validators and future proposed upgrades towards accessible and scalable upgrades.

The thesis presents the missing steps in understanding the direct link between these protocol upgrades and their heavy usage on the underlying and more modern networking layer. We provide directives, open-source tools, and detailed analysis of how and how much the current leading peer-to-peer networking literature and state-of-the-art impacts relevant aspects of upper applications such as centralization of nodes and profitability of validators.

1.2 Challenges

The research of peer-to-peer networks in the blockchain and web3 landscapes presents a series of evolving challenges that directly impact the development, deployment, and optimization of decentralized services. This section explores the complex challenges faced in this ever-changing field, highlighting the importance of a nuanced approach to the natural development of these technologies.

One of the primary challenges comes from the rapid evolution of the Web3 space and blockchain technologies. This fast growth rate of development often surpasses the ability of developers, users, and regulatory frameworks to keep up, leading to a gap between technological capabilities and practical applications.

Furthermore, the emergence of various edge protocols, applications, and platforms supporting decentralization introduces complexity in choosing the optimal solutions for specific use cases. While this diversification fosters innovation, it also complicates the landscape, making it challenging for newcomers or technology adopters.

A notable concern within this ecosystem is the predominant focus on the development of new protocols or the upgrading of existing ones, generally relying on simulations rather than real-world applications. This trend overshadows the critical need to monitor the performance and identify bottlenecks of current protocols, thereby impeding their organic evolution. The absence of experimental measurements on actual implementations poses a significant obstacle. In many instances, the lack of established methodologies and open-source tools for this purpose exacerbates the challenge, making it difficult to assess and enhance the protocols' performance genuinely.

Another intricate issue is the difficulty of bridging the gap between the networking layer of peer-to-peer (P2P) applications and the applications' performance. This complexity arises from the decentralized nature of these applications, where performance metrics and optimization strategies are not as straightforward as in traditional centralized systems.

Lastly, the absence of a validated or homogeneous methodology for evaluating and improving the decentralized web compounds these challenges. Without a standard approach, efforts to optimize and evolve the decentralized web can be disjointed and less effective.

In conclusion, addressing these challenges necessitates a concerted effort from the community to prioritize the evaluation and optimization of existing technologies alongside the exploration of new ones. Establishing standardized methodologies for performance assessment and fostering an environment that encourages the open sharing of tools and data can significantly contribute to the organic evolution of the Web3 and blockchain ecosystems.

1.3 Objectives

The constant evolution of Ethereum's platform is a complex community endeavour to generate a leading decentralized finance (DeFi) platform that offers robust technology, a secure and decentralized governance model, and a rich ecosystem of applications. The fast and constantly evolving platform proposes and aggregates many new changes and upgrades on its journey, making it hard to correlate network and protocol behaviours with the causes that so many upgrades could have originated. The objectives are two-fold. On the one hand, the objective is to study the software logic, hardware, and network topology updates that the PoW to PoS transition originated; on the other hand, it is to correlate how these three aspects interact with each other, and understand how these interactions interfere and limit in some cases the performance and future upgrades of the protocol. Below, we discuss the objectives in detail.

The first objective of the thesis is to provide a profound overview of the changes that the PoS upgrade of the platform has meant at three primary levels. The transition promised a vast reduction in the hardware requirements, banishing block mining, which considerably reduces the carbon footprint of the platform and allows the participation of a more extensive set of users in the network. Thus, the objective is to measure the actual requirement changes, the impact on the network's topology, and the new proposed incentive system. We aim to provide novel and previously nonexistent methodologies and tools that can provide this information in real-time, which could indicate the health of local nodes and the overall network.

The second objective is to empirically identify how these apparently distinct layers depend on and interact with each other. Characterizing network and hardware interdependencies, network and performance direct links, or hardware with performance correlations could be crucial to spot or monitor the protocols' current limitations or their implementations. Furthermore, a deep understanding of these interconnected layers and their effects on the network and its participants can help us predict or reduce important implications such as hidden centralization incentives or possible attack vectors. The third objective is to explore how the current PoS specifications and the peer-to-peer state-of-the-art literature might somehow limit the platform's resilience and scalability. As data sharding was targeted as the scalability-enabler solution in Ethereum's roadmap, the objective is to explore how it could be successfully implemented using existing, mature enough, and battle-tested tools and protocols. Through validated simulations, we explore how well a Data Availability Sampling (DAS) scheme could be implemented in Ethereum's PoS.

1.4 Contributions

In this section, we offer an overview of the principal contributions within this thesis. The contributions are delineated across five distinct and sustained chapters, each corresponding to one or a set of papers published in peer-reviewed journals and

conferences. Combined, these chapters deliver some novel methodologies and studies to measure each of the three main layers: software logic, hardware, and the p2p network, as well as the challenges and solutions in the convergence of the same ones.

Chapter 2 gives a background of peer-to-peer networks and the main existing protocols, the main components of blockchains, and a survey of Ethereum and its transition towards a PoS-powered consensus. This chapter is essential as it presents and digs deep into the fundamental concepts required to understand the following thesis chapters, allowing the document to be self-explanatory. The chapter presents some prior related work, which is extended further in the following chapters.

Chapter 3 presents the network topology and the network insights of Ethereum's consensus layer. The chapter introduces the first network crawler for the network and digs into the healthiness of the peer discovery Distributed Hash Table (DHT) and the overall active and contacted nodes over time. It includes a discussion of the different identified distributions, such as the geographical concentration of nodes or the client diversity adoption of users in the network. In the analysis, we introduce the network's main communication channels and the centralization vector that these ones can imply.

Chapter 4 presents the hardware resource requirements reduction that implementing PoS meant for Ethereum. We first list and introduce the software infrastructure needed to participate in the network along with the six main available software to participate in the chain's consensus. We then present the multiple end-users that choose to run a node in the network and the ideal hardware target to do so. We thoroughly analyse the main hardware usage differences across the different software and hardware platforms. Furthermore, we provide some observations that help current and future users estimate the infrastructure more accurately according to their needs.

Chapter 5 presents a deep background on the consensus specification and proposed reward or incentive system in Ethereum after the merge of the consensus and the execution layers. We present the direct link between the validator's duties and the profitability of the same ones, modelling a performance indicator that compares the reward of a validator on a given epoch with the theoretically Maximum Extractable Reward (MER) it could have gotten. This performance indicator is presented and used to compare the performance across validators or any set of validators. The chapter finishes by analyzing how we used this methodology, benchmarking the performance of some staking entities before and after the merge.

Chapter 6 is the core of this thesis and provides the link between the previously presented chapters. The chapter first introduces the strict slot time organization that Ethereum's PoS specification defines. It introduces how this narrow window of action can actually impact the performance of validators, where the difference of a second could define the difference between successfully performing the validator's duties and getting rewarded, or missing that chance and getting penalized. The chapter presents two main studies where the time restriction could play an important role. The first one introduces how network latency can put a node that is not that close to the core of the network in an unfavourable position because messages take more time to reach. The second one measures and explores the viability of DVT as a staking software alternative that brings deployment resilience, where the extra time of aggregating validator signatures could exceed the strictly defined time windows.

Chapter 7 is the last contribution chapter of the thesis, and it is the most exploratory of the previous four. The chapter presents the scaling roadmap of Ethereum, introducing one of the major limitations of the proposed data-sharding schema,

DAS. The chapter explores the aggregation of a Kademlia-based DHT to fulfil the existing DAS challenge through the usage of a simulator. We present the existing limitations of these networks, validating our results and the overall simulator results with one of the most popular DHTs, the IPFS one. The chapter describes why these DHTs are not entirely suitable for solving the puzzle and describes the pros and cons of the currently available solutions.

In Chapter 8, we summarize the main contributions of this thesis and delve into the background efforts that culminated in this work. Additionally, we outline potential directions for future research. All the tools developed and introduced in this thesis are open-source and freely accessible to the research community. This accessibility represents another significant contribution of this thesis, promoting and following the collaborative spirit of distributed networks, web3, and blockchains.

Chapter 2

Background

In this chapter, we present a detailed background of the evolution of peer-to-peer networks, the impact of Ethereum in the web3 ecosystem, and the insights of some of the core components around the topic that are relevant to understanding the remainder of this thesis.

2.1 Peer-to-Peer Networks

Peer-to-peer (P2P) networks represent a decentralized paradigm in computer networking [67, 150], revolutionizing how digital interactions occur and information is disseminated. Emerging from the fundamental human desire for collaboration and sharing [102], P2P networks eliminate the need for centralized servers, allowing direct communication and resource sharing among interconnected nodes. Since the inception of P2P networks, exemplified by early applications like Napster [28], they have evolved into versatile infrastructures supporting various functionalities such as file sharing, content distribution, and distributed computing. Despite their widespread adoption, P2P networks pose unique challenges in scalability, robustness, and security due to their decentralized nature [10]. Understanding the underlying principles and dynamics of P2P networks is crucial for unlocking their full potential and addressing these challenges effectively.

2.1.1 Journey of Distributed Networks

The origins of peer-to-peer (P2P) networks relate to the late 1990s, with the apparition of pioneering applications like Napster [18] and Gnutella [166]. Napster gained immense popularity as one of the earliest file-sharing platforms, enabling users to share music files directly with each other. Its centralized server maintained an index of available files, facilitating search and download operations. However, Napster's centralized architecture made it vulnerable to legal challenges, leading to its eventual shutdown due to copyright infringement issues [28]. Following Napster's demise, decentralized P2P networks emerged as a response to the limitations and vulnerabilities of centralized systems [102]. Gnutella became the first decentralized network that enabled peer-to-peer file sharing without reliance on central servers. It's architecture allowed users to connect directly with each other, forming a distributed network where search queries and file transfers occurred peer-to-peer [122].

As P2P technology continued to evolve, new generations of decentralized networks emerged, each addressing specific challenges and adapting to diverse use cases. One notable example is BitTorrent [151], which introduced a protocol for distributed file sharing. It relied on a swarm-based approach, where users collaborated to share files by downloading and uploading fragments of data simultaneously [209,

107]. This decentralized approach not only improved efficiency and scalability but also mitigated the reliance on individual servers for file distribution.

In parallel with advancements in P2P file sharing, the concept of decentralized computing gained traction, leading to the development of blockchain technology. Bitcoin [21, 131] pioneered the use of blockchain as a distributed ledger to record transactions securely and transparently. Blockchain's decentralized architecture, coupled with cryptographic principles, ensured trust and integrity in peer-to-peer transactions. This compiled a solid basis to support decentralized applications beyond cryptocurrencies.

Today, P2P networks continue to evolve and diversify, with emerging technology upgrades, the standardisation of protocols and libraries such as Libp2p, the apparition of decentralized content addressing platforms as the InterPlanetary File System (IPFS) [17], and blockchain platforms that enable smart contracts and decentralized applications (DApps) [27] like Ethereum [208]. These innovations underscore the enduring significance of peer-to-peer networking principles in reshaping digital interactions, information exchange, and collaborative computing in the modern era.

2.1.2 Definition and Characteristics of P2P Networks

P2P networks have several fundamental characteristics that differentiate them from traditional client-server architectures. Firstly, these distributed networks promote a decentralized architecture, where each peer has equal status and autonomy, enabling direct communication and resource sharing between peers without intermediaries. This decentralized nature fosters self-organization within the network, empowering peers to manage and maintain the system's operation and integrity collectively [160].

Distributed networks heavily rely on peer and content discovery, as well as on routing mechanisms, which play a vital role in optimizing resource allocation and enhancing network efficiency. Furthermore, these networks have promoted the apparition of algorithms for locating resources, maintaining overlay structures, and optimizing message routing, which are essential for minimizing latency and maximizing throughput in P2P networks [103, 66].

Due to their constantly evolving nature, peer-to-peer networks are resilient and fault-tolerance systems [203], which dynamically adapt to sudden node failures or departures [184]. To achieve such an operation level, they support strategies such as data replication, redundancy, and distributed storage mechanisms, which ultimately maintain data integrity and accessibility even in the presence of adverse conditions or malicious activities.

In summary, P2P networks embody a decentralized and collaborative approach to information exchange and resource sharing, characterized by self-organization, reliability, security, content discovery, incentive mechanisms, and legal considerations. Understanding these fundamental characteristics is essential for designing and managing P2P systems effectively in various application domains.

2.1.3 Challenges to Overcome

Although P2P networks have many benefits and advantages, as previously introduced, not everything is that shiny. The operation of these ones must coordinate multiple users and service providers without a central authority that has sovereign control over the network, which undoubtedly complicates the process making it less optimal than its direct counterpart.

Given the distributed and open nature of the system, security and privacy are critical concerns in P2P networks [145, 13]. Although distributed protocols heavily rely on cryptographic techniques to ensure encrypted communications and peer authentication protocols, these networks still publicly expose users to the Internet at some point. Despite more modern networks and protocols trying to mitigate these attack vectors, they could still be subdued to data tampering or malicious attacks such as Sybil attacks or eclipse attacks [98, 202].

It is not new that distributed networks suffer from scalability issues, particularly concerning accommodating many peers while maintaining performance and efficiency [10]. The presented challenges include managing network topology to ensure robustness and fault tolerance, optimizing data routing to minimize latency, and mitigating issues related to network congestion and bandwidth limitations. Due to the heavy usage of redundancy and replication techniques to overcome the network dynamics [66, 174], the extra network overload of having to process and distribute duplicated data considerably reduces the processing capabilities of unique events.

Although we won't dig deeper into this topic over the thesis, peer-to-peer networks also face legal and regulatory challenges [106], particularly concerning copyright infringement, intellectual property rights, and data protection. Network operators must navigate legal frameworks and comply with regulations governing content distribution, privacy, and liability to ensure lawful operation and mitigate legal risks. This heavily impacts the network capabilities

There has been incredible work from the research community to provide meaningful insights and improvements on distributed peer-to-peer networks. Especially after the "boom" of these ones with the apparition of blockchain and cryptocurrencies, we've presented the sudden spike in measurements and protocol optimizations over the existing literature. Although, from our understanding of these platforms, they combine the evolution of p2p networks and the blockchain and consensus advances, most of the existing literature independently addresses the limitations of both topics. In fact, there aren't many relevant works trying to match the state, the healthiness and performance of the network with the healthiness and operation of the application layer.

2.2 Blockchain

Over the past two decades, distributed applications have surged in popularity as an alternative approach to providing decentralized web services [100], countering the trend of mass centralization across internet services. In an era marked by ubiquitous and low-cost internet access, distributed systems empower users to engage actively as both consumers and providers of services. This proliferation of service hosts fosters a resilient infrastructure less reliant on any single point of control compared to purely centralized systems. By distributing the system among participant nodes, the strength of decentralized applications hinges on the collective participation of each peer, thereby decentralizing control away from any singular authority [67].

2.2.1 Prime on Blockchain Technology

Blockchain is a continuously growing public ledger, where using cryptographic techniques, storage units or blocks are constantly concatenated to the blockchain [143]. Primarily employed in decentralized applications, this public ledger meticulously records and monitors every network event, encapsulating them within those blocks.

In such decentralised ecosystems, these networks rely on predefined rules or consensus mechanisms to determine the next block proposer [105, 14]. Block proposers, typically under a certain degree of randomness, propose and append a new block to the blockchain. However, their participation is often incentivised by the rewards that adhering blocks that follow the established rules generate. Regardless of the proposer's intentions, each proposed block undergoes thorough validation and verification by the entire network before being incorporated into local chains and, consequently, the public ledger.

Active users within the network typically maintain a locally accessible copy of the blockchain, which is crucial to validate the latest network events, especially for those aspiring to become the next block proposer. This involves keeping up to date with the newest blocks and network occurrences, commonly called tracking the chain's canonical state or head of the chain, thereby ensuring synchronization with the latest public block.

However, certain anomalies may arise during the life-cycle of a blockchain protocol, such as a sudden decrease in active validators or network inconsistencies, leading to disagreements among participants. These discrepancies can result in a temporary chain split, known as forks, where a secondary or side head of the chain emerges [134, 180]. The lack of consensus on event verification often attracts a portion of active users to these forks. Stability is restored when the network realigns, with most participants consenting to follow a specific fork, disregarding the others.

2.2.2 Reaching Consensus

Permissionless blockchains achieve decentralization by adopting open-access policies, ensuring participants can operate without a central authority governing them [126, 145]. In the best-case scenario of a distributed network, all participants would adopt an honest contributing behaviour. Nonetheless, in reality, the participants' anonymity and the general economic value attached to the tokens generated in the protocols originate from a non-trusted environment surrounding the protocol. In such a hostile environment, the compiling components of the network and the protocol are exposed to failure due to malicious actors, as these can spread imperfect or misleading information that can interfere with the correct operation of the network. The described situation corresponds to a Byzantine Fault-Tolerance [55] situation and showcases the need to establish rules to coordinate the block proposals and unanimous block validations in the network. This compilation of rules is recorded under what is called a consensus mechanism [205]. The unification of these methods together with the local copy of the chain that participants keep provides a homogeneous criterion for participants to judge or validate the occurring events, granting to the network the capacity to achieve the following:

- Validity (Correctness): New proposed blocks, following protocols, must be accepted and stored by any honest node in the network.
- Agreement (Consistency): An honest node adopts a block header into its chain view once confirmed by another node.
- Liveness (Termination): Transactions from honest participants are eventually confirmed.
- Order: All nodes accept the same transactions as long as they align with their local blockchain views.

As a result of adopting the exact consensus mechanism at the definition of blockchain protocol specifications, the network knows how to propose and validate blocks regarding the non-trusted original environment. Ironically, the technology provides certain trust levels in the *chaos* of the internet, where anonymous network peers accept and follow the agreed consensus rules, contributing to the security and health of the network.

In the case of blockchain technology, the consensus rules are applied over each of the new proposed blocks. However, to ensure the immutability of content previously aggregated into the chain, as displayed in figure 2.1, these networks rely on concatenating blocks linking the hash of the parent or previous block into the next ones' content. Relieving the consensus mechanism to regulate which is the legitimate next block proposer and the criteria to define whether the block is valid or not.

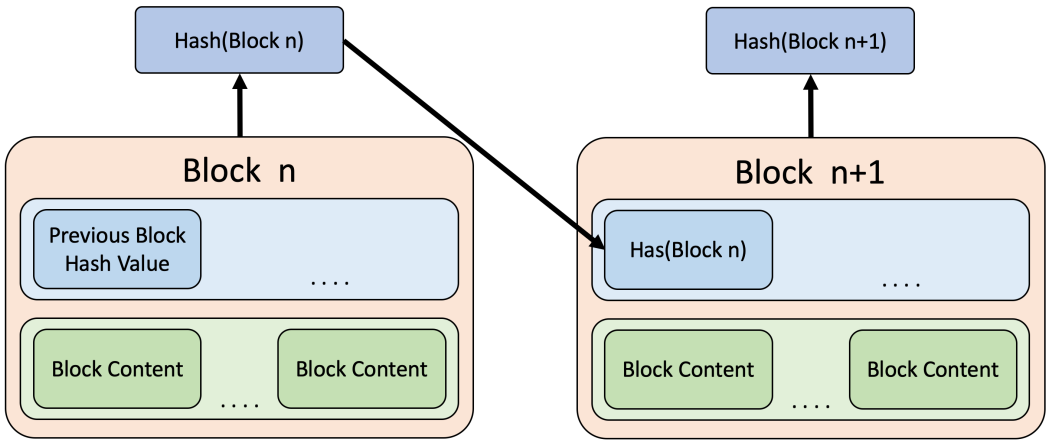

FIGURE 2.1: Block concatenation in blockchain technology.

It sounds flawless on paper; however, as previously mentioned, the consensus mechanism can not fully rely on the honesty of participants. Thus, each consensus actor correctly following the rules is rewarded as part of the blockchain protocol and the consensus mechanism. Generally interpreted as economic remuneration, the received chain tokens incentivise the honest following of the rules [68]. This token remuneration makes an honest performance more profitable than an attack on the platform.

2.3 Ethereum

Ethereum [208, 62] 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) [80] and its dedicated programming language Solidity [207]. With its proprietary programming language directly integrated into the platform, the platform serves as the basis for users to create or define arbitrary custom rules for ownership or to create transaction formats and state transaction functions. This way, Ethereum was the first decentralized platform to offer the freedom/possibility to write smart

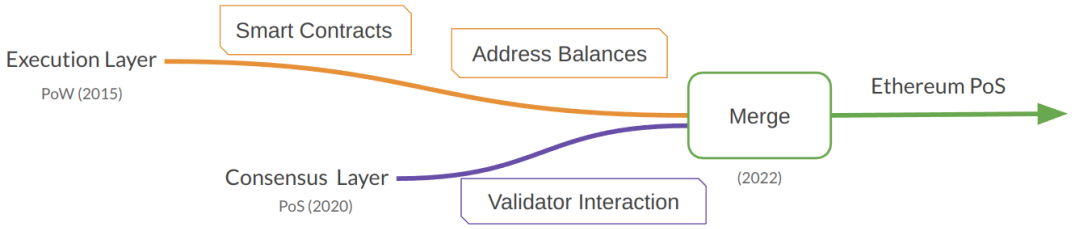

FIGURE 2.2: Ethereum transition from PoW to PoS.

contracts and decentralized applications. The smart contracts and decentralized applications are the Ethereum equivalence for Bitcoin transactions, with the difference that they are the compilation of operations that can generate different functionalities.

With a consolidated peer-to-peer network and its capabilities to process *Smart Contracts*, it currently handles above 1 million transactions per day from 600.000 active accounts [138]. 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 being 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 [190] among its participants. However, as the very first blockchain ever presented to the world, Bitcoin [132], Ethereum shared a similar set of limitations in terms of scalability and sustainability [76].

Until its recent transition to PoS introduced in the following section 2.3.1, it proposed an alternative reward system for nodes in the network performing the block, transactions, or smart contracts processing. It introduced the famous Gas term to regulate and reward validators on the platform. Understanding Gas as the cost for users for executing in the *EVM*, the cost of each running operation gets defined by the computational resources of the operation that the transaction and block validator have to dedicate. To display the amount of Gas that each process demands, the platform describes *Gas Price* and *Gas Limit* as terms to calculate and determine the fees obtained by validators[1].

- Gas Limit: The maximum number of Gas that the entire operation may consume.
- Gas Price: The price that the operation requester is willing to pay per each Gas unit *wei*, generally expressed in *gwei* (10^9*wei*).

2.3.1 Ethereum's Transition to PoS

As long as the network proved to be resilient, powering the consensus through PoW required massive and unnecessary power consumption. As one of the most significant upgrades in the platform's lifetime, Ethereum embraced a long journey to shift its consensus mechanism from PoW to a more sustainable Proof of Stake (PoS) [172]. This transition reflects the hard work and dedication of the blockchain research community, which was consolidated with the deployment of Ethereum's Beacon Chain in December 2020 [29] on its *Phase0* [146].

Following the initial plan of combining both, the figure 2.2 shows how the two coexisting PoW chain and the young beacon chain fusion into a single blockchain

[1]The following Gas division presents the initial reward system for processing transactions. After London's hardfork and the application of *EIP-1559*, this changed; check section 5.3 for further details.

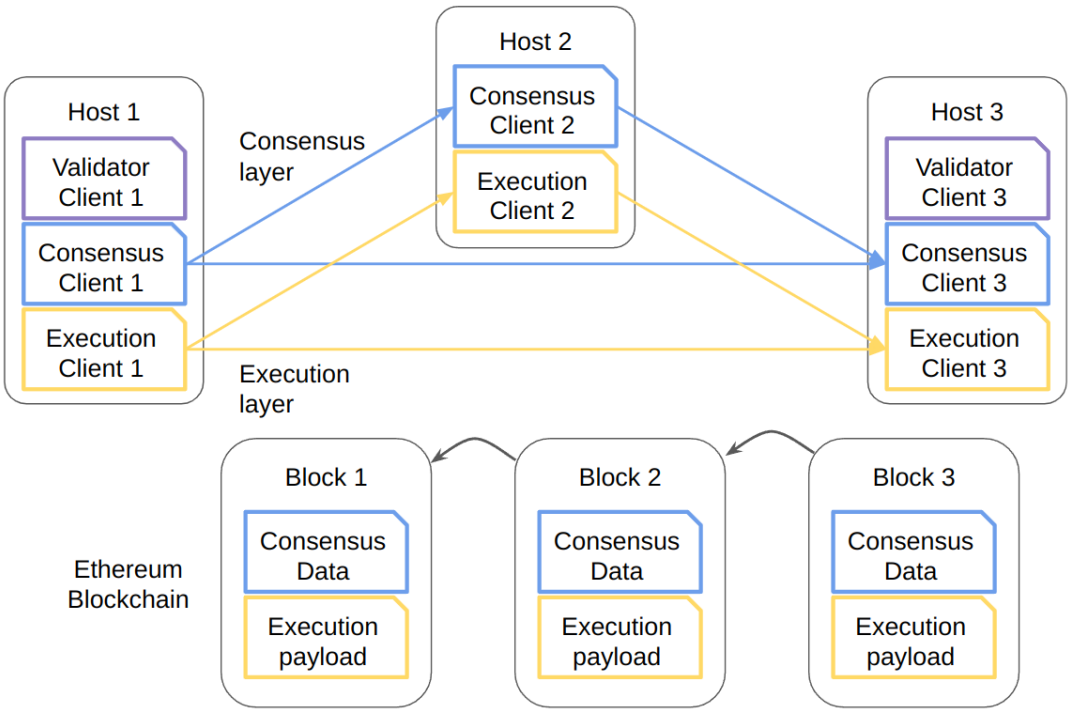

FIGURE 2.3: Subdivision in layers of Ethereum after the merge.

with "The Merge". After this event, Ethereum's consensus was finally achieved through PoS and the interaction between validators. However, the network remains split into two separate layers that need each other, as figure 2.3 shows: the Execution Layer (EL) and the Consensus Layer (CL). The first one still tracks, validates, and executes the transactions that users share to the network, including the processing of the *Smart Contracts* [207] through the Ethereum Virtual Machine (EVM) [82]. On the other hand, the second one, the CL, defines, captures, and processes the interaction between validators in their attempt to reach consensus through Gasper FFG for finalization [26], LMD GHOST [129] as a fork-choice rule and RANDAO [64] as RNG to assign the following duties. As figure 2.3, Ethereum now needs several software to run: i) an Execution Client that operates in the Execution Layer and executes block payloads (transactions, smart contracts, etc.), ii) a Beacon Client or Beacon Node that operates in the Consensus Layer and decides which is the canonical chain with the rest of the network, and iii) a Validator Client that only communicates with the beacon client to sign the needed duties with the validator's private key.

The consensus layer of Ethereum is organized in epochs. Each epoch contains 32 time windows of 12 seconds called *slots* where a single validator elected from the RANDAO Reveal algorithm has the chance to aggregate a new block to the beacon chain. Since the rest of the existing active validators must reach a consensus over each proposed block, splitting the epoch in 32 slots helps reduce the computational load of processing the duties of 900.000+ (at the time of writing the Thesis) active validators[2]. Thus, the whole list of validators is divided into the 32 slots and then into a maximum of 64 committees (check figure 2.4 for reference). This way, each added block serves as the main unit of time where new historical data is added to the beacon chain. Therefore, each block includes validators' attestations through aggregations, which, due to the low frequency at which validators need to produce them, are one of the primary performance and steadiness indicators for analyzing

[2]https://ethseer.io/

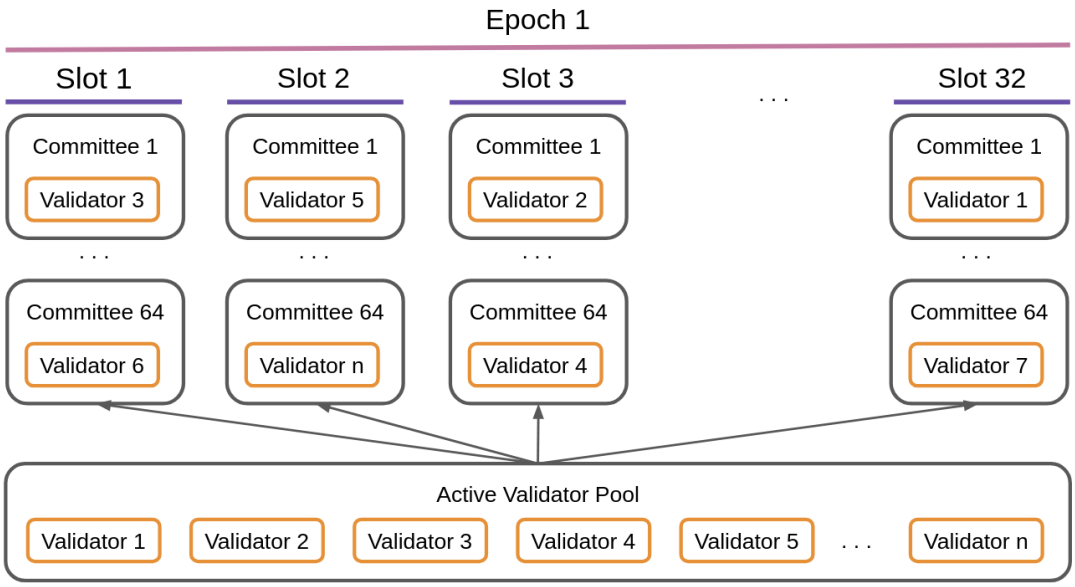

FIGURE 2.4: Assignation of active validators into attestation committees.

and comparing the performance of a validator. This leaves us three primary duties describing a validator's participation and profitability in Ethereum's PoS.

Attestations Each validator in the CL is assigned to a specific slot and committee per epoch. Furthermore, it must contribute with votes that later on are collectively aggregated before their inclusion in subsequent proposed blocks. This aggregation process significantly decreases block size by efficiently tracking validator duties and streamlines the process for future block proposers, who only need to consider the most recent aggregations from each slot. As these contributions or votes create the immutable links between the blocks, the votes have three primary factors, which also determine the 'quality' of their attestation and, thus, the reward of the validator:

- **Source:** This is the hash of the justified checkpoint when the proposed block was proposed. In the context of the CL, checkpoints represent the Beacon State root at the start of an epoch, encompassing the state transition outcomes from the preceding epoch.
- **Target:** This refers to the hash of the epoch's initial block.
- **Beacon Block Root**: This is the block's hash being attested to.

Validators have 32 slots within which to produce these attestations. A second crucial parameter influencing the quality of an attestation is the 'inclusion delay'. This delay denotes the number of slots that pass before an attestation and are included in a block after the one attested to. Optimal validator performance is achieved when a vote is cast with all three flags correctly identified and included in the next block, resulting in an inclusion delay of just one slot.

Block Proposals A designated active validator creates and submits a beacon block in each slot. At this juncture, the validator integrates the necessary metadata and incorporates the maximum number of aggregated attestations, subject to a cap

of 128 per beacon block. The compensatory reward for the proposer is contingent on both the quantity and quality of these attestations. Specifically, for each attestation flag not previously included, a portion of the generated reward is allocated to the proposer of the block containing it. Consequently, incorporating more novel attestations directly enhances the proposer's reward.

Similarly, the proposer benefits from including sync committee duties, receiving a share of the overall reward they produce. This arrangement motivates the block proposer to include the maximum number of sync committee duties, maximizing potential rewards.

Sync Committees Sync committees represent the consensus of a small group of validators over the "Root" of the head Beacon State. This helps light clients validate blocks without fully downloading and processing the beacon chain. Each sync committee comprises 512 randomly selected validators who sign new block headers every slot and rotate every 256 epochs (8192 slots). However, they typically aren't classified as performance indicators due to the low frequency at which a validator participates in a sync committee.

2.3.2 Protocol Defined Slot Time Ranges

We've already delved into the temporal segmentation of Ethereum CL's blockchain. However, as figure 2.5 shows, the protocol still defines smaller time subdivisions inside each slot. These numbers are just guidelines. However, following them is vital for an optimal network operation. The figure summarizes the tasks that need to be performed in the following order:

1. First, the elected block proposers must create and broadcast a new block at the beginning of the slot (second zero). This provides 4 entire seconds for the message to propagate over the participants in the network. This means that the block proposer had 4 seconds before the start of the slot to receive and group aggregated attestations from the previous 32 slots.

2. After the first 4 seconds of the slots, validators assigned to attest to it are expected to generate and broadcast their votes with their perception of the chain (attesting to the *source*, *target*, and *head* they see). They share this vote with the corresponding beacon committee aggregators, and the spec assigns the same time range of 4 seconds to broadcast the message.

3. Finally, committee aggregators must collect votes between seconds 4 to 8, producing the aggregated attestations at the 8th second of the slot. In all committees, 16 validators are randomly selected to aggregate and broadcast the attestations. After that 8th second, the network disposes of 4 extra seconds so that the next block proposer has enough time to receive all the aggregations.

Keeping the correct timing between these tasks is crucial to avoid confusion in the network. For example, if a block proposer extends the creation of its block for 10 seconds, the block could be received later than 12 seconds since the start of the slot, risking being voted as a missed block. If a validator waits too long to generate and send the attestation, the aggregators might not include that vote in the same slot, increasing the inclusion delay and reducing the final reward.

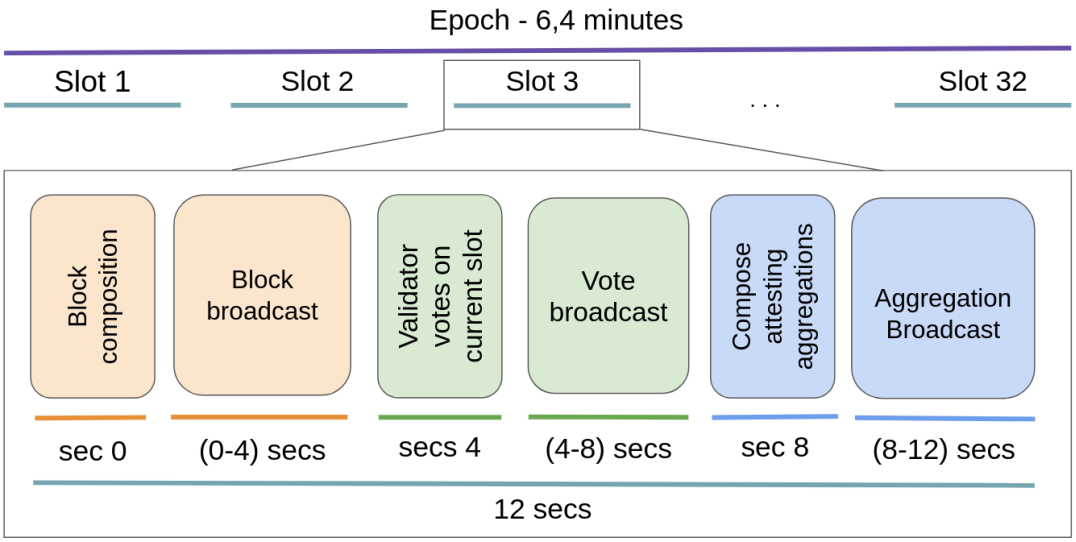

FIGURE 2.5: Expected temporal division of a slot, including the time each duty should be sent and propagated over the network.

2.3.3 Ethereum validators and testing networks

Protocols like Ethereum are complex in many aspects. They are hard to consolidate at the technical and human levels; they have too many components and contributors. From that basis, organising the possible upgrades of the protocols is even more complex, proposing changes that need to be done, reviewed and tested. Like many other projects in the web3 ecosystem, Ethereum adopted a basic methodology of splitting all the ideas into features or Ethereum Improvement Proposals[3] (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.

Since Ethereum's launch in 2015, it has had many EIPs and testnets. This testing environment represents an isolated environment where a few specific changes, events, interactions, 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 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. They are stress-tested to find the weak points of the protocol or the implementation. Figure 2.6 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 and its prior test in the Goerli/Prater testnet. It is necessary to notice that, since public testnets such as Goerli[4], participation is not monetarily incentivised, participation tends to be lower than on the main networks. This might not sound that critical if it stays above the 66%. However, this can slightly affect the steadiness and profitability results of validators participating in the network.

[3]https://eips.ethereum.org/
[4]https://goerli.net/

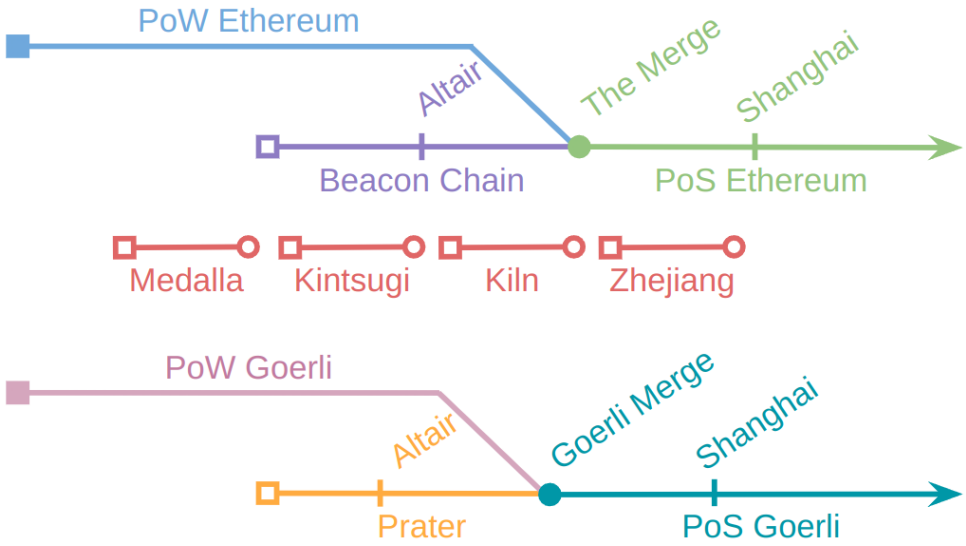

FIGURE 2.6: Ethereum network's hard-forks organization, public testnets in red, Goerli testnet at the bottom. Note that hardforks are tested on testnets before they occur on mainnet.

2.4 Networking protocols

Due to the complexity of managing and coordinating distributed networks, peer-to-peer networks lie on essential protocols that ensure efficient, secure, and reliable participant data exchange. Among the most significant "must" protocols of any p2p network, we can find discovery protocols responsible for locating peers within the network and data transfer protocols, which facilitate the actual exchange of data. Together, these protocols empower P2P networks to support various applications, from file sharing and distributed computing to blockchain technologies and beyond, providing a robust framework for decentralized communication and collaboration.

2.4.1 Libp2p

Libp2p is a modular network stack and framework for building peer-to-peer network applications. It is designed to be language-agnostic, allowing developers to use it in various community maintained programming languages such as Go[5], Rust[6], Java[7], JavaScript[8], or Nim[9]. It provides a set of protocols, libraries, and tools that enable decentralized communication, peer discovery, and secure data transfer among nodes in a network. The main key features and components that Libp2p provide are:

1. **Modularity**: It's built with a modular architecture, allowing developers to select and use only the necessary components for their specific use cases. This modular design promotes flexibility, interoperability, and code reuse across P2P applications.

2. **Peer Discovery**: It offers multiple peer discovery mechanisms, including multicast DNS (mDNS), peer routing protocols (such as Kademlia and distributed

[5]https://github.com/libp2p/go-libp2p
[6]https://github.com/libp2p/rust-libp2p
[7]https://github.com/libp2p/jvm-libp2p
[8]https://github.com/libp2p/js-libp2p
[9]https://github.com/status-im/nim-libp2p

hash tables), and custom discovery mechanisms. These mechanisms enable nodes to discover and connect to other peers in the network dynamically.

3. **Transport Agnosticism**: It abstracts the underlying transport layer, allowing developers to choose from a variety of transport protocols (such as TCP/IP, UDP, WebSockets, QUIC) based on their requirements and network conditions. This transport agnosticism enhances interoperability and adaptability across different network environments.

4. **Security**: It incorporates cryptographic primitives and protocols for secure communication, including encryption, authentication, and message integrity. It supports various cryptographic algorithms and key exchange mechanisms to ensure the confidentiality, integrity, and authenticity of data exchanged between peers.

5. **NAT Traversal**: It includes built-in support for network address translation (NAT) traversal techniques, such as hole punching and relay services, to facilitate peer-to-peer communication across firewalls, routers, and other network obstacles.

6. **Protocol Multiplexing**: Libp2p enables concurrent communication over multiple protocols by implementing protocol multiplexing. This allows applications to use different communication protocols simultaneously without interfering with each other, improving efficiency and resource utilization.

Overall, Libp2p provides a robust foundation for building decentralized and resilient P2P networks, making it a popular choice among developers working on distributed systems, blockchain platforms, decentralized applications (dApps), and other peer-to-peer network applications.

2.4.2 GossipSub

To better understand GossipSub [198], we must introduce the PubSub protocol. PubSub is a P2P protocol implementation that obtains a rapid distribution and propagation of messages through a network using the publisher and subscriber method [15].

On a PubSub network, peers get organized in meshes, one for each of the message topics, leaving every peer with as many peer meshes as message topics subscribed. The name of the protocol is very descriptive; when a publisher peer (sender) wants to propagate a message on a given topic, it sends the message to the peers that subscribed to the same message topics, finished on its mesh. Later, when subscribers receive the message from the peer mesh, they adopt the role of publishers and distribute the message to the peers from its topic mesh. This cascade effect provokes a fast and total message propagation through the network, achieving the lowest possible latency on message transmission and making PubSub suitable for decentralized applications.

GossipSub is a version of the PubSub protocol designed to distribute messages efficiently while conserving network resources. It introduces the concept of mesh degrees, allowing peers to customize their connectivity in each topic meshes, reducing bandwidth usage and the overall network load. GossipSub adds the perioding gossip of seen messages. These exchanges of message metadata help spread information through a "lazy push" method while increasing security against Sybil attacks. The protocol dynamically manages peer connections and topic subscriptions through graft and prune messages, adjusting the mesh network based on activity. It also validates messages to optimize bandwidth and prevent spam, making

it suitable for low-resource devices. GossipSub has been heavily adopted by modern decentralized networks such as Celestia[10], Filecoin[11], Polkadot[12], or Ethereum's Consensus Layer, highlighting the sustainability and optimization that the protocol can achieve in such networks that focus on providing scalable solutions to the web3 ecosystem.

2.4.3 Ethereum Node Records (ENR)

Ethereum Node Records (ENR) [61] are crucial components of Ethereum's peer-to-peer (P2P) networking protocol. ENRs represent the unique identifiers for Ethereum nodes within the network. Each node maintains its own ENR, which contains information about its network address, capabilities, and other relevant metadata, such as the capabilities and features supported by the Ethereum node.

They are crucial in bootstrapping new nodes into the Ethereum network and facilitating peer discovery. New nodes can use ENRs to identify and connect to existing nodes, thereby joining the network and participating in Ethereum's decentralized consensus mechanism. These records include cryptographic signatures to ensure their authenticity and integrity. By signing their ENRs with their private keys, Ethereum nodes can demonstrate ownership and authenticity, preventing spoofing and tampering attempts.

ENRs are designed to be dynamically updateable, allowing Ethereum nodes to update their records in response to changes in their network configuration or capabilities. This flexibility enables nodes to adapt to evolving network conditions and requirements, allowing peers to filter out peers based on their latest metadata.

2.4.4 DHTs

Distributed Hash Tables (DHT) are decentralized data management systems that provide data storage and retrievability across a distributed network of nodes. These systems are popular because of their capability to spread data across many different nodes efficiently. Hence, each node is responsible for storing a small portion of the data without relying on a single location for data storage. DHTs achieved this decentralization by using a hashing algorithm to assign each piece of data a unique key, determining where the data should be stored within the network. The following key components generally characterize them:

- Decentralization: with no central controlling node, all nodes cooperate to form the network, making the system more resilient to failures and censorship.
- Scalability: DHTs can generally handle adding or removing nodes with minimal disruption, making it easy to scale the system up or down.
- Efficiency: Retrieving or storing data involves communicating with a portion of the available nodes rather than the entire network, leading to efficient use of resources.
- Fault Tolerance: Data is often replicated across multiple nodes, ensuring that if some nodes fail or leave the network, data is not lost and remains accessible.

[10] https://celestia.org/
[11] https://filecoin.io/
[12] https://polkadot.network/

Kademlia DHT

Kademlia [123] is a specific type of DHT, popular for its simplicity and efficiency in peer-to-peer networks. It introduces several innovative features that optimize the process of node lookup and data storage in a decentralized network. Kademlia uses an XOR metric to define the distance between two nodes or between a node and a key. This distance is not geographical but a logical distance that compares the hash of the node identifiers (nodeIDs) or between a node and a data key. This logical link ensures that each node can efficiently route queries to the node most likely to have the desired information.

Nodes in a Kademlia network organize themselves in a structured overlay network, where each node maintains a list of contacts sorted into "buckets" based on their distance from the node. This structure enables efficient routing of queries by progressively forwarding a query to nodes closer to the target key.

The efficiency of the protocol comes from parallelising the search process with concurrent queries to multiple nodes, which significantly speeds up the lookup process. A node looking for a specific key or another node will contact the closest nodes it knows and iteratively refine the search until it locates the target. This iterative approach significantly reduces the number of hops needed to find a node or data.

2.4.5 Discovery5

As a decentralized platform, Ethereum tries to avoid any point of centralization inside the proposed protocol. The networking area, specifically the peer discovery, is no exception to this rule. The implemented GossipSub protocol, despite being protocol-oriented to message propagation on decentralized applications, does not offer any peer discovery service, at least not a fully functional one[13]. Leaving that task in charge of the application. To overcome the existing lack of such an important service, the core developers designed a dedicated discovery protocol for Ethereum nodes, Discovery 5[52] (*discv5*), currently on its 5.1 version [70].

Discv5 focuses on the Ethereum Node Records (ENR) exchange along with the peers. The ENRs of the nodes present in the network, which include the needed information to establish a connection, are recorded in a DHT, a Kademlia DHT, to be precise. This DTH offers the possibility to sample and search along with the generated database, allowing it to easily update the node record of a specific peer if modifications are detected. As an entry point to the *discv5* protocol, the nodes can search for others that publicly advertise themselves on particular topics. As an alternative that improves the performance, the first connection with boot-nodes[14] is possible.

The peer discovery service is essential for the network in general. New peers joining the network need connections from where to synchronize the chain. Peers already on the network also need time to update their information source, ensuring they correctly follow the chain's head. Furthermore, to increase the network's resilience to Sybil attacks, *discv5* serves as a record of nodes on the network that the node could connect at any point.

[13]GossipSub offers a functionality that allows directly connected participants to exchange peers when one of them drops the connection.

[14]Boot-Nodes are super light publicly available nodes whose purpose is to offer the first list of participant nodes to those new users in the network.

2.4.6 The InterPlanetary File System (IPFS)

IPFS [17] distributed platform for storing and retrieving content-addressable media objects. IPFS is a community-driven and open-source project, with significant contributions from over 1185 code contributors across 400+ organizations, including universities, startups, and large corporations [194]. ProtocolLabs[15] plays a vital role in supporting IPFS, contributing most of the codebase and funding most full-time contributors. However, the project emphasizes community involvement in decision-making through public voting. IPFS has achieved widespread adoption, with over 3 million web client accesses and over 300,000 unique nodes serving weekly content [12]. It supports various decentralized web applications, and its usage has been extended to some popular browsers like Opera and Brave.

Key points of its success are: i) its content-based addressing, as it allows content to be served and retrieved from any peer in the network, dynamically adjusting the content routing to alterations in the network, ii) its decentralized object indexing, IPFS uses a peer-to-peer overlay network for redundantly indexing all available object locations, reducing dependency on any single point of failure, iii) its immutability, objects in IPFS are self-certified through cryptographic hashing ensuring the verifiability of the downloaded content, and iv) its open participation, as anyone can join and contribute to the IPFS network without needing special permissions.

[15] https://protocol.ai/

Chapter 3

Fundamentals of Ethereum's P2P Network

3.1 Introduction

The transition of Ethereum towards a PoS consensus brings significant changes to the protocol at many levels: at the hardware level, at the networking level, and obviously at the consensus level. On this major protocol update, Ethereum aims to create a more sustainable protocol, enabling the foundations for the future scalability of the same one simultaneously. The thesis has already presented the changes on the first and the second levels in the previous chapter 2. Thus, this chapter sets the basis for addressing the changes at the networking level.

This transition had several effects on the networking side. The first and clearer one relates to the complexity of the underlying peer-to-peer network, as the new approach compiles two coexisting networks. The PoS approach refers to them as two separate layers: the Execution layer, which broadcasts and processes the interaction with the EVM, and the Consensus layer or Beacon Chain, which tracks all the interactions between validators while they reach consensus. Each network keeps its protocols, handshakes, and message types, requiring two separate clients to participate, as shown in figure 2.3. Thus, each network has its nodes, topology, and characteristics.

The second effect on the network is related to the size of this one. With lighter hardware requirements to participate in the network, the community expects to see a more significant number of nodes in the network. This increase in the number of nodes impacts, at the same time, the main aspects of the peer-to-peer network, such as the topology of the same one or the message broadcasting time within the same one. A larger network brings more resilience [109] to the network, providing a broader range of fault tolerance towards a sudden peak of peer disconnections. However, everything comes at some cost, and larger networks mean more delays upon message deliveries when broadcasting a new block or an attestation to the network.

Under a constantly evolving protocol, as new EIPs and changes are continuously implemented at each hardfork or protocol upgrade, there is wide uncertainty about how this network is organized. With such a variety of software options to participate on the network, it is still unknown which is the distribution of software usage or the version update adoption over time. Furthermore, since the beacon chain has adopted the Libp2p (refer to section 2.4.1) networking protocol stack, a proposed standard of libraries and protocols for the distributed and peer-to-peer networks, the networking conditions and the interoperability across clients remain unknown.

3.2 Background

While Ethereum's network topology remains unknown after adopting PoS, the topology, the characteristics, and the behaviour of peer-to-peer networks have been extensively researched over the last two decades. Since the apparition of the first distributed network, Gnutella [166], some of those works [122, 177, 90] pointed out that distributed networks, by nature, tend to self-organize through the usage of methods or protocols like DHT [31]. Such protocols help on important topics like peer discovery [141] or service discovery [148]. This provides the network with unique properties and layouts that can directly affect the performance, reliability, scalability, and sometimes anonymity of the applications that rely on them. The common take of these works focuses on the importance that the topology of the overlay network has on the performance of the network. Thus, they propose an active approach of participating in the network as a user for digging and mapping the topology of these chaotic networks.

On the other hand, as discussed in [185], the size of these networks constantly grows as the size of the information that gets shared, leaving the previous approaches and methods outdated. Authors showcase that measuring the network from a single actor's point of view lacks a better perspective of the network's total. Thus, they propose a non-intrusive technique to track the available nodes over time passively, the traffic volume, and the host distribution in the network.

With the popularization of blockchain technology over the last decade, new distributed protocols based on blockchain technology have appeared, e.g., Bitcoin and Ethereum. The impact of these decentralized protocols has augmented in the industrial and academic fields as shown in [128], triggering back remarkable attention on the peer-to-peer networks they rely on.

Due to the heavy reliance of these protocols on distributed peer-to-peer networks, previous works like [47, 49, 204] already provided compelling insights from the networks of the most prominent Blockchain protocols, Bitcoin and the former PoW Ethereum. In these works, the authors show methods to track the number of nodes on the overlay, dig into the peer discovery over the network, and even some light message exchange of information among the peers. Due to the recent network update on Ethereum's network, this chapter of the Thesis analyses the Ethereum main network performed with an open-sourced network crawler.

The proposed approach leverages some previous ideas from [16, 99, 20], such as the use of network crawlers to exploit possible network weaknesses that might be a hazard to the integrity of the protocol. As explained in those papers, the information about the connections between the network nodes is as important as the information about the nodes them-self. Thus, we provide a study on the consensus layer's network, including the analysis of:

- The discovered topology of the network overlay.
- The distribution between the client types available to participate in the network.
- The distribution of nodes hosted on cloud providers.

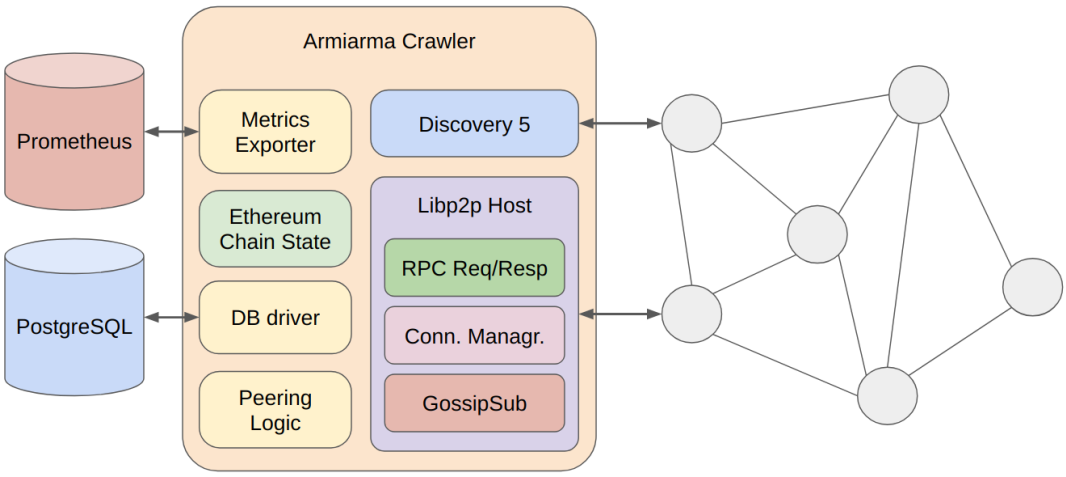

FIGURE 3.1: Diagram of the Armiarma crawler's internal structure and its interaction with the network.

3.3 Methodology

As previously introduced, there has been a lot of research on finding the best approach for measuring a distributed network's topology and characteristics. The existing literature suggested that to achieve an optimal and realistic representation of the network's overlay, the combination of active participation and passive measurements is the best existing solution. This could be done through a network crawler that participates on the network for a defined period, which enables proactively testing the connectivity of the discovered nodes while providing enough time to receive passive connections for those peers behind firewalls or under Network Address Translation (NAT) tables.

Due to the lack of existing network monitoring tools in the Libp2p and Ethereum ecosystem, in aggregation to the "peakiness" of the Ethereum beacon chain's connectivity, we developed a light Ethereum network node that can discover, interact, and record metrics from the network.

3.3.1 Network Crawler

Armiarma [6], the Ethereum network crawler, is the software in charge of discovering and connecting peers in the network. It replicates the essential networking modules and protocols, behaving like a light networking node. The following paragraphs present the main parts of the network crawler, which are also presented in figure 3.1.

Discovery5 As introduced in section 2.4.5, Ethereum relies on a public Kademlia-based DHT and a protocol called Discovery5, or discv5, for peer discovery. The protocol allows nodes to advertise their ENRs 2.4.3 with the networking information to establish a connection. Furthermore, the constant request of random nodes to the network allows the discovery of newer peers. This protocol is vital to reach the most significant part of the network. Thus, the crawler supports and contributes actively to the peer discovery process of Ethereum. To increase the desirability of being connected with the crawler when composing the crawler's ENR, the crawler supports all the Ethereum consensus layer's metadata, including the advertisement of the subscription of all the "attestation_subnets".

Libp2p Host Once multiple ENRs are discovered from discv5, the Ethereum specification[1] defines that the communication between nodes needs to be done through the Libp2p connections and streams. The crawler does create a Libp2p host to do so. It composes a Libp2p "Peer Identicator", or "peerID", from a new Secp256k1 [19]-EDCSA [89] private key pairs that Ethereum's specification require, which is also the same private key pair used for the crawler's ENR.

Therefore, the network and the crawler try to keep connections open with nodes to track any occurring events. These events are received through the message broadcasting protocol GossipSub. As previously introduced in section 2.4.2, GossipSub is a PubSub routing protocol that can efficiently share information packages over a peer-to-peer overlay. Since keeping track of the latest news (new blocks or validators' attestations) has a significant impact on the nodes' performance, the crawler supports the GossipSub routing protocol, being able to contribute and track all the relying traffic on the network.

Geographical Location of Nodes The crawler relies on a third-party API to locate the operation placement of the nodes in the network. The crawler includes a module that connects to the IP-API[2] free tier API, respecting the current limitation of 45 requests/second. This allows the crawler to extract vital and updated information such as Internet Service Provider (ISP), the country, the city, whether the node is hosted on a data centre, and much more metadata from the advertised IP.

Chain State We've already mentioned that Ethereum connectivity is tricky. Establishing connections requires specific patterns that can define whether the connection is accepted. And some of those requirements are related to the node's perception of the chain. As peers could belong to any of the existing different Ethereum networks (check section 2.3.3 as reference), not all the connections are "useful" for each node. Thus, the specification requires a chain status exchange during the connection handshake.

Although the remote peer is identified during the first Libp2p handshake, Armiarma's ability to join and track GossipSub topics directly benefits from its capabilities of maintaining connections. Thus, the crawler must keep a local state of the network to be accurately shared during Ethereum's handshake.

Exporting Metrics The crawler's main objective is to extract and store information from nodes in the network. This information is generally obtained over the discovery of ENRs, the connection handshakes, or over the message broadcasting topics. However, all the information has to be stored somewhere. The crawler has two separate metrics sharing points. The first one is a PostgreSQL [130] database that keeps the raw data per peerID, nodeID, and IP. It offers some extra relational tables for storing the connection events, the arrival of GossipSub messages, or even a snapshot of the active peers under a defined frequency. This could be understood as the raw data the crawler generated while operating. The second one is a Prometheus [3] time-based database, which keeps all the aggregated metrics the crawler updates and exposes every 15 seconds. Among these metrics, the crawler exposes internal debugging metrics such as the number of ENRs discovered on the last day, the number of connections per GossipSub topic, the client distributions, and so on.

[1] https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md
[2] https://ip-api.com/
[3] https://github.com/prometheus/prometheus

Crawler's Configuration The crawler was developed with various configuration properties to create a multipurpose crawler that crawls multiple networks. Among these options, any user can configure the networking details (IP and Port), the alias that the crawler will use in the network, the endpoint of the database that will be used, the port at which the debugging metrics will be exposed, the desired network that wants to be crawled, etc... These configuration options ensure optimal interaction between the crawler and the network, keeping track of a single chain and a single network.

3.4 Evaluation

The complete version of the crawler, version "v2.0.0", has been running on an OVH cloud server with limited resources to replicate a low-computational power environment. The machine hosting the crawler has 4 CPU cores and 8GB of memory, and it has been up and running since March 17th, 2023. The following subsections describe the data and the progression of the metrics gathered by the crawler about Ethereum's network.

3.4.1 Ethereum's Public DHT

The network's security in blockchain protocols relies on keeping many nodes collaboratively working together. However, due to the distributed nature of the projects, none can ensure a 100% uptime from all contributors. This makes resilience a key component of the network. In the case of blockchains such as Ethereum's current PoS implementation, the network needs to provide participants a safety margin so they can go offline without necessarily leading to a protocol crash. In this case, Ethereum needs at least 66% of the validators actively attesting blocks to finalize [25]. We've already talked about the internal structure of how an Ethereum node is run, where no consensus node maps to one validator. Each node in the consensus layer can host from zero to thousands of validators without giving any evidence of how many validators are hosting. However, since the network that helps broadcast or synchronize blocks and attestations is composed of beacon nodes, having an overview of how many nodes participate on the network is relevant.

The default method to discover participants in the network is through the Discovery5 protocol, which enables any node to find new participants. Maintaining the same connections over a long period can have its counterparts, as relying only on steady connections increases the risk of Sybil attacks or eclipse attacks, which can ultimately derive to leaving a node on a chain's fork [3].

For that reason, keeping an eye on Ethereum's public DHT is important to monitor the total size of the network and its healthiness. Figure 3.2 shows the number of unique ENRs actively shared with the crawler using the discv5 protocol over the last six months before writing this chapter (16th of February 2024). The figure shows that the crawler could discover a mean of 69.533 active ENRs on Ethereum's mainnet daily.

However, receiving an ENR over the discv5 protocol doesn't mean the node is online or connected. The discv5 protocol assigns a local Time To Live (TTL) of 24 hours to each node it interacts with, so the protocol contemplates the existence of non-reachable nodes, or as they are described in the DHT literature, stale records in the DHT [183]. Because the crawler tries to establish a direct connection with each of the nodes it discovers through discv5 using a Libp2p connection, we were able

FIGURE 3.2: Number of ENRs actively shared with the Crawler in the last 24 hours. Snapshot from the 16th of February 2024.

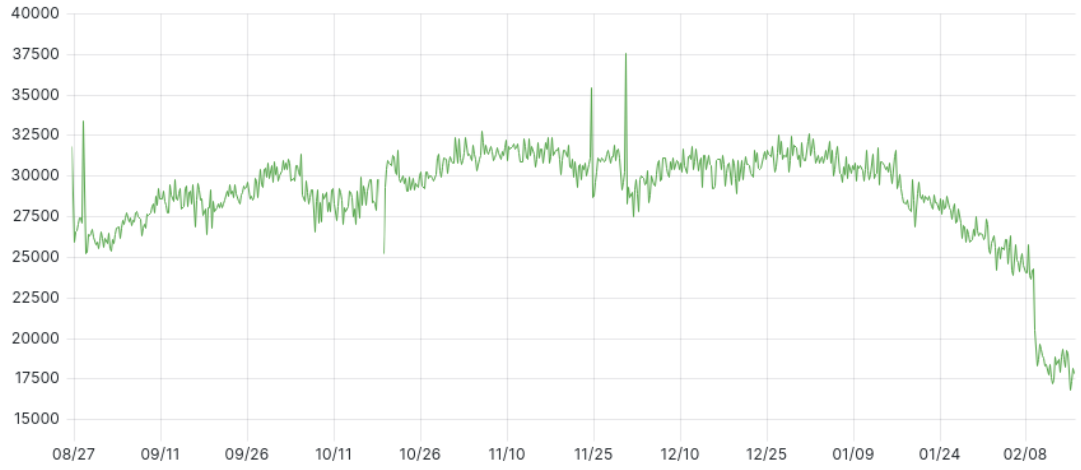

FIGURE 3.3: Number of ENRs that are not deprecated. Nodes with ENRs were actively shared on the discv5 protocols, plus the nodes we could connect with within the last three hours.

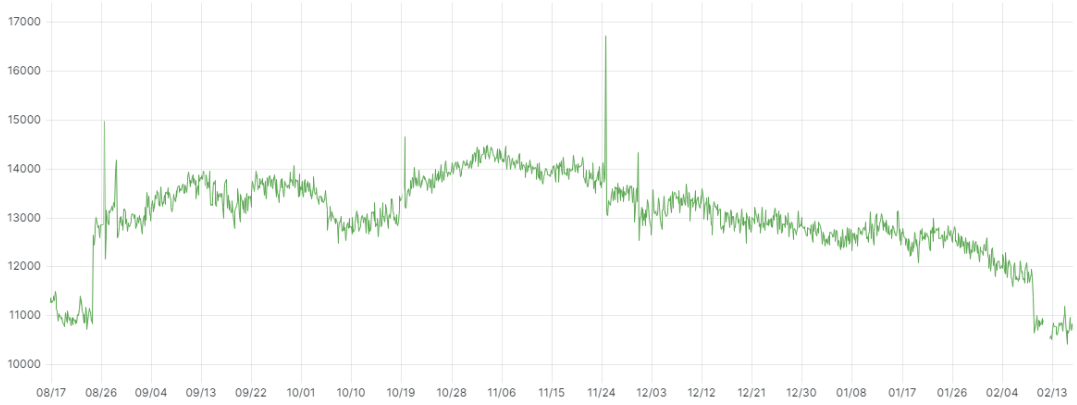

FIGURE 3.4: Number of active nodes in the Ethereum Network. Nodes that weren't deprecated at the snapshot time and that were successfully connected and identified at least once.

to filter out those nodes that are not reachable, or that could have already left the network. Figure 3.3 shows the left peers we could connect over the last three hours or those whose ENRs were seen in the last three hours. Every node that doesn't fulfil the following description is what we, the crawler, identify as "deprcated", denoting that a peer is no longer eligible to be online. The filter shortens the mean of 69.533 discovered ENRs to the mean of 28.960 non-deprecated peers per day.

If the result of the Libp2p connections is successful, and the crawler can identify the remote peer by its "user agent", we can have the certainty that the peer is actively participating in the network. This aggregation of filters allows us to compose the graph of peers shown in figure 3.4. The figure shows the distribution of the resulting daily active nodes, where we can appreciate that the number gets reduced even more, reaching the daily 13.119 nodes.

We expected many peer dialling rejections or encounters hidden behind NAT or no longer connected nodes. However, the reduction of 69.533 ENRs to 28.960 non-deprecated peers to only 13.119 shows that the connectedness in Ethereum's peer-to-peer network is not that great. Taking a deeper look at what could be the reason behind this network behaviour, we identified that the result of each connection attempt hides more information than we initially thought. Figure 3.5 describes the distribution of errors that the non-deprecated nodes reported on the last connection attempt, which at the date of 16th of February 2024 were 17.852 nodes. The figure shows six main replies where the connection timeout leads on the most common reply with 7.376 nodes reporting it. It is closely followed by the successful connection, shown in the figure as a "none" error, with 5.193 peers connected without any issue. The rest of the errors are what we categorize as occasional errors, which include the failure to negotiate the security protocols, a failure on the dialling due to a taking too long Libp2p identify protocol, a direct refuse of the connection by the remote peer, or the sudden close of the stream opened with the remote peer.

As previous works [177, 31, 96] point out, the base of some p2p networks is likely to rely on a small portion of peers from the entire network. These peers compound the network's nucleus, often called "core-peers" or "super-peers". These are peers with whom it is easy to connect and transmit information reliably. The crawler identified 5.193 peers that were reachable without issues, sharing and exchanging data easily. These peers could represent a good portion of that "nucleus" of Ethereum's CL network. In this case, we can appreciate how validators' economic incentive to

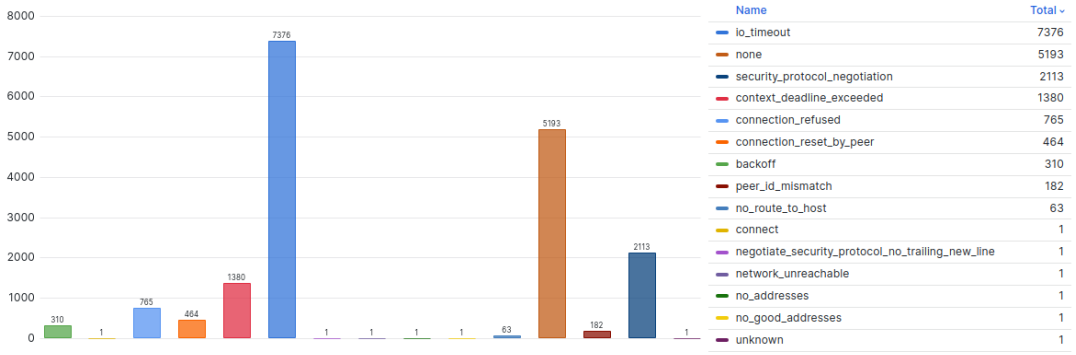

FIGURE 3.5: Error distribution of the individual connection attempts to non-deprecated peers. Snapshot from the 16th of February 2024.

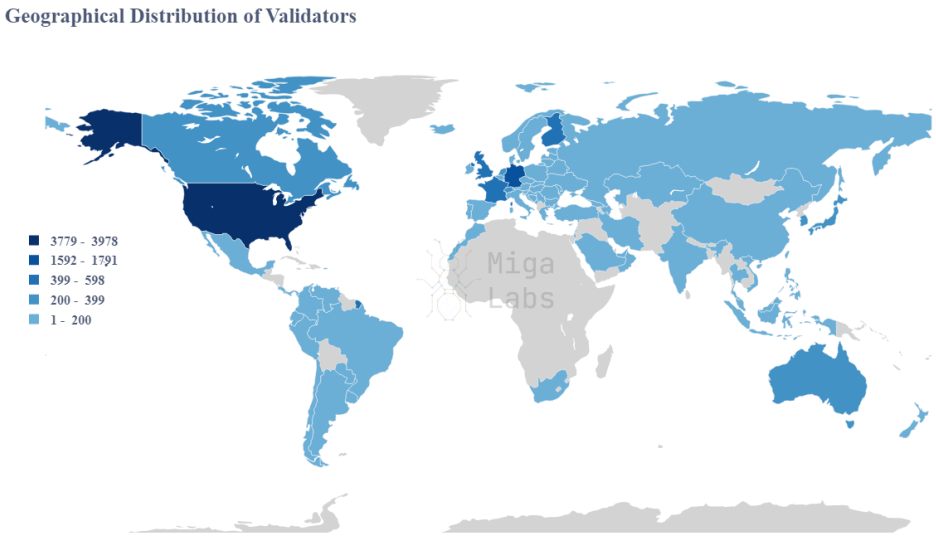

FIGURE 3.6: Geographical distribution of the Ethereum active nodes. Snapshot from the 16th of February 2024.

stay online directly impacts the connectivity of peers at the network's core.

3.4.2 Network Metrics

The distributed nature of blockchain networks [113] has a noticeable impact on the network's overlay [84]. These are geographically distributed worldwide, and dispersed clusters of nodes reduce the likelihood of being affected by the same type of regional problems or events (e.g., national censorship). Figure 3.6 presents the geographical distribution of the nodes tracked by the crawler at 16th of February 2024. The figure showcases that the network is spread around the six main continents, with 92 different countries hosting at least one Ethereum node. However, the figure also showcases a heavy concentration of nodes in just a few countries, as almost half of the peer distribution concentrates in two countries, the US (37,52%) and Germany (15,67%).

In a decentralized network, low latency connections are essential for tracking every occurring event as fast as possible. In this case, on the beacon chain, every 12

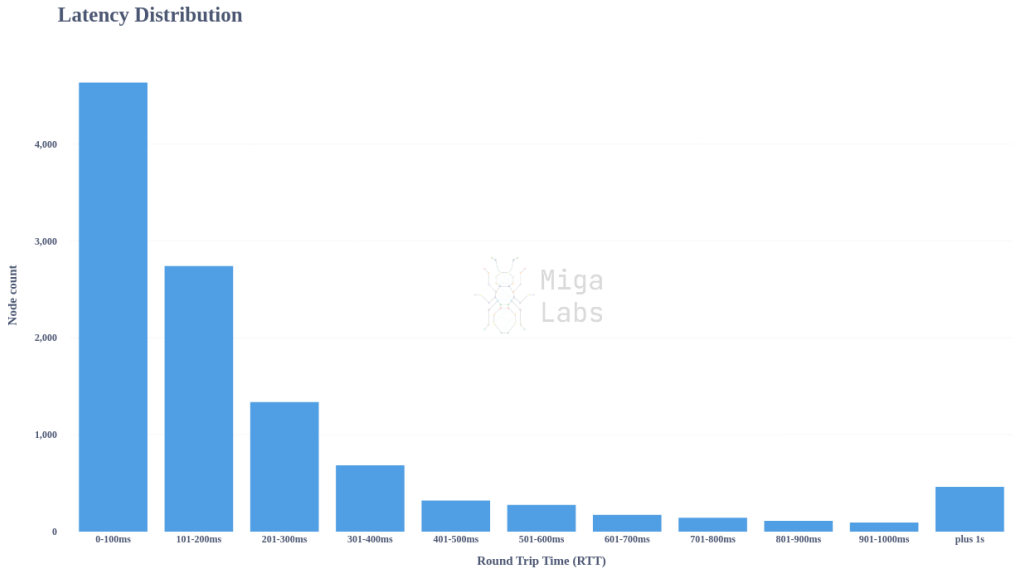

FIGURE 3.7: Distribution of Round Trip Time (RTT) between the active nodes and the crawler. Snapshot from the 16th of February 2024.

second, a randomly chosen validator will have the chance to propose a block and receive a generous reward. In figure 3.7, we can observe the Round Trip Time (RTT) of the latest connections the crawler recorded while connected to the peers. The RTT measures the time it takes from sending a request to the destination peer until we receive a response. The latency between the destination peer and us is half the RTT or lower. We can observe that 90% of the peers have an RTT below the 500ms, having only a 3.4% of the nodes reporting an RTT over the second. Note that the crawling process was done from a machine in Frankfurt and that the RTT includes the time the requester took to process the request from Libp2p's Identify protocol.

The apparition of large staking entities and specialized companies in the operation of nodes has impacted the distribution of nodes over the peer-to-peer overlay. Large operators must rely on multiple nodes to host all the validators, as it reduces the risk factor of keeping validators offline from a node malfunctioning. Thus, splitting the validators across multiple nodes in the setup is a common practice to minimise risks and increase resilience.

Figure 3.8 shows the clusterization of nodes per IP. The figure can be read as follows, the X axis shows the number of nodes that an IP could host, while the Y axis shows the number of IPs that host that many nodes. The figure shows how the 82.24% of the Ethereum nodes run on a single IP, which corresponds to the 91.95% of the listed IPs. This has a counterpart, though. As each validator participates on a given slot's committee every epoch, nodes generally advertise the attestation subnet subscriptions on their ENRs. This increases the capabilities of nodes to find peers in the GossipSub topic they need for optimal performance. Figure 3.9 shows that exact distribution, showing the number of active nodes that advertise which number of subnets. These numbers suggest that a large part of the active validators and, thus, the network, is highly concentrated on a small set of nodes. The figure shows that 90% of the nodes concentrate on zero to two attestation subnet subscriptions. It could look like only 16.6% of the nodes don't host any validators. However, since

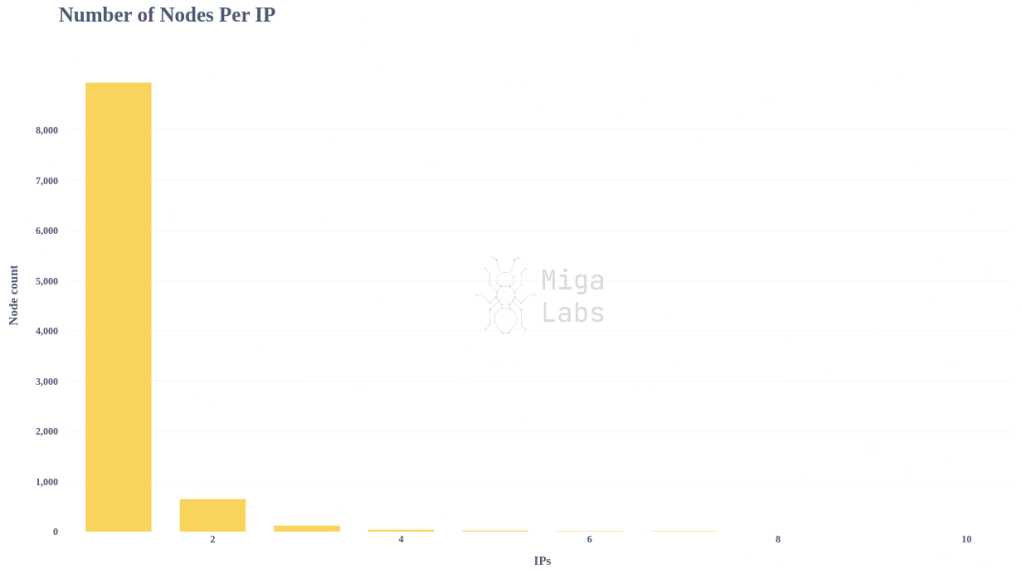

FIGURE 3.8: Distribution of IPs shared across the active nodes in the network. Snapshot from the 16th of February 2024.

the apparition of the "Deterministic Subnets" proposal[4], the vast majority of nodes now subscribe to at least two subnets even if they don't host any validator. The proposal aimed to use non-validating nodes beneficially to the network, sacrificing a bit of bandwidth for a healthier GossipSub mesh.

However, the network doesn't only concentrate validators on a small set of nodes. Scaling and managing the hardware might also become a problem for those large entities, and not all of them might be willing to pay the extra cost of buying and running it. Thus, the network has significant participation from data or cloud services such as AWS, Hetzner, or OVH. Figure 3.10 shows the categorization of the IPs that were identified by the crawler, where we can appreciate how only those three cloud machine providers account for the 15.42%, 12.34%, and 8.9% of the nodes, respectively.

3.4.3 Diversity

Despite the network seeming to be centralized, to some degree, geographically and to the service of some data server providers, the network has many other aspects where diversity can be a health indicator for the network. As chapter 4 will present in further detail, there are five to six official software[5] to choose from when participating in the network.

Due to Libp2p's Identify protocol, when the crawler manages to establish a connection with a node in the network, it can track down the "user agent" that the node uses to identify itself in the network. Most clients expose their name on the client together with some extra metadata such as the version and even, in some cases, the architecture of the binary used or the OS that it was compiled for. Figure 3.11 shows the client distribution of the active nodes in the Ethereum network. The figure shows a big improvement since the first report done at [37], where Prysm alone was

[4]https://migalabs.io/blog/post/observing-deterministic-subnets-in-the-wild

[5]Some users only consider the five open-sourced options as the main options, as Grandine, to the date of writing this Thesis (16th of February 2024) remains close-sourced.

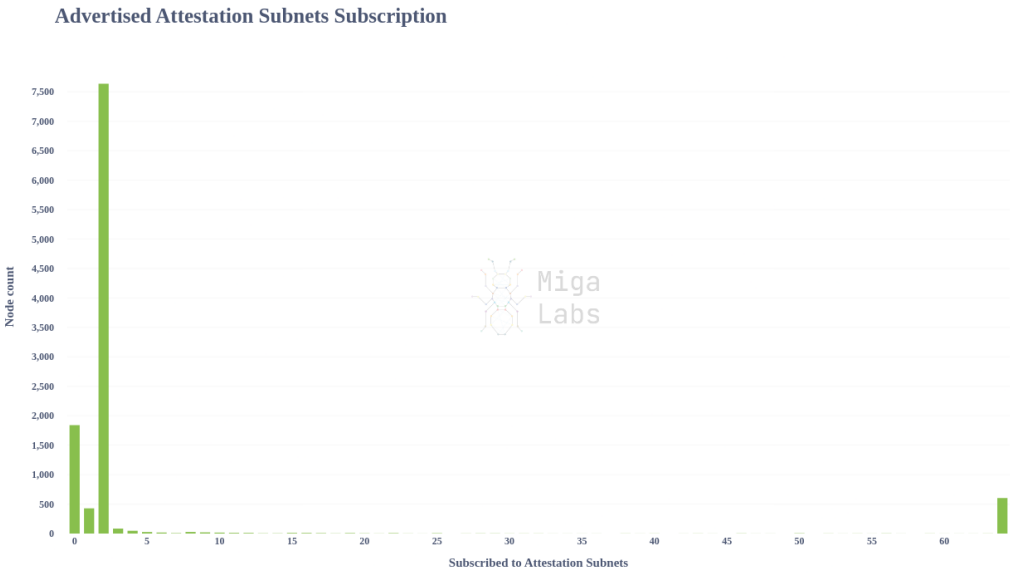

FIGURE 3.9: Distribution of the advertised number of attestation subnets' subscriptions from active nodes in the network. Snapshot from the 16th of February 2024.

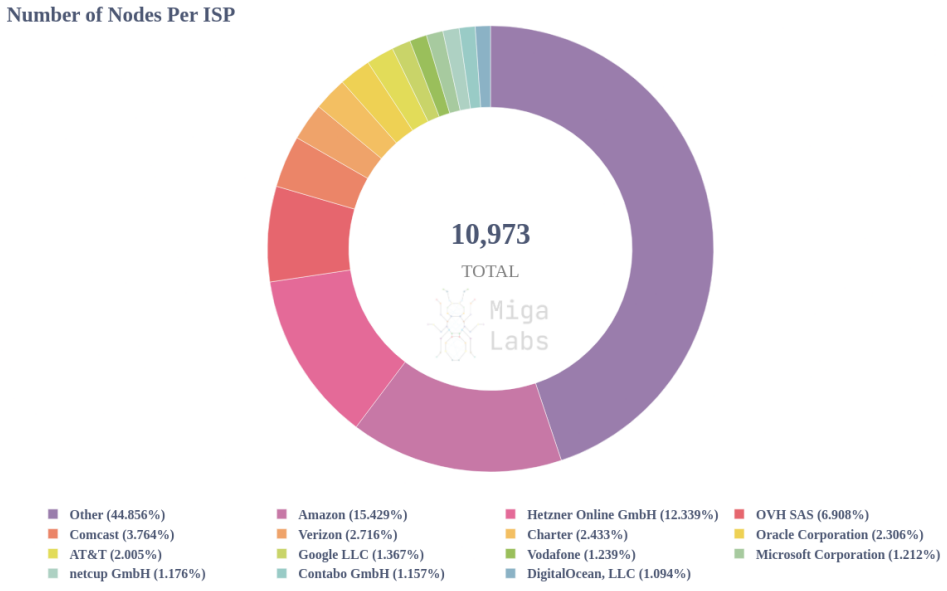

FIGURE 3.10: Distribution of different ISPs for the active nodes in the network. Snapshot from the 16th of February 2024.

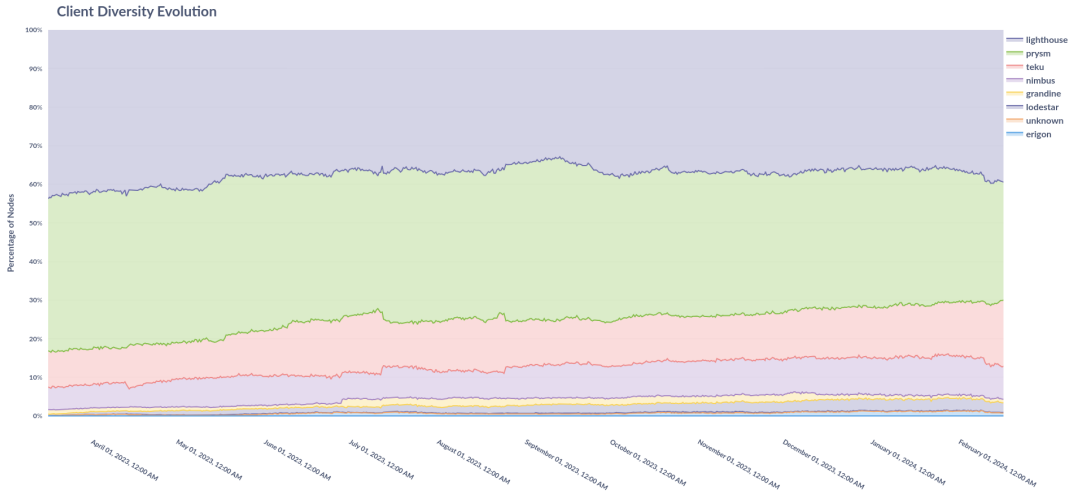

FIGURE 3.11: Distribution of client software used by active nodes in the network over time.

reported to be almost the 83% of the network in 2021. After some community effort and awareness, the distribution has become evener. The network still relies heavily on Lighthouse and Prysm, which currently keep a 39.3% and 30.4%, respectively. On the other hand, Teku and Nimbus follow up with 17.1% and 7.8% of the network share. Unfortunately, we still see that some clients struggle to get acceptance from the network, which leaves Lodestar and Grandine with the remaining 2.4% and the 0.9% of the nodes.

3.5 Conclusion And Future Work

This chapter of the Thesis presents the observed geographical and client distribution of Ethereum's beacon chain. We found that although the platform incentivizes decentralization for security purposes, decentralization is not entirely achieved yet. We have identified that almost 53.2% of the almost 11,000 online nodes in the network are located in the US and Germany, representing the opposite of a geographically decentralized network. Previous works [97] have demonstrated that the networking performance completely relies on the location of the node. This means that nodes in regions far from the mentioned two countries could be at a disadvantage, which could in turn lead to some level of geographical centralization.

In addition to the geographical distribution issue, the presented work shows that the network has been able to react to a previous critical mono-client dependency. The presented documents show that the biggest share of the network relies on the two single clients, which reduces the chances of vulnerability due to a bug in a single particular client. This is one of the most critical points this crawler analysis highlights because a bug impacted that specific client on April 24th 2021 causing finalization issues during multiple hours [57].

We've presented the evidence that a significant part of the network is running on data centres. Furthermore, we could speculate on a big share of active validators running a small part of the network. Despite the network has shown important improvements in client diversity, increasing also the resilience of the network, there are still many aspects to improve.

Chapter 4

Evolution of Hardware Requirements with Ethereum's PoS Transition

4.1 Introduction

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 [48], 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 [121], the more obsolete the previous hardware becomes. This highly incentivises the acquisition of newer mining devices, indirectly increasing the network's overall energy consumption [110] 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).

This chapter 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.

4.2 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 must invest to achieve 51% of the mining power [71]. 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 in Ethereum, where the network's total hashing power has constantly increased over time [140, 187, 201]. Many efforts in the research and industrial communities were dedicated to optimising and speeding up the process of hashing [182, 2]. Furthermore, the direct application of those hashing methodologies on mining [83] defined a rapid hardware evolution in the field. Starting from the usage of GPU accelerating techniques [50, 179], and reaching, in some cases like Bitcoin, the development of custom ASICs [121].

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 [34]. But the transition to PoS adds more complex logic to the existing protocols, as this mechanism presents a few critical points that could potentially be exploited.

Prior research works have tried to consolidate blockchain technology in the Internet of Things (IoT) field [164, 178]. 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 [169].

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 [69] and 13.000 for the young PoS beacon chain, as chapter 3 introduced. 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 [115, 135]. The diversity concept has been widely used in many industries like the military and the aerospace ones [114, 196], providing resilience even to computer networks [211]. 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 [23]. Code diversity in the blockchain space has been previously analyzed [163], 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 [88]. 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 [137]. 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, 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 [171], in this chapter, 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 chapter 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.

4.3 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[1], Nimbus[2], Lighthouse[3], Prysm[4], Lodestar[5], and Grandine[6], 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 chapter 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

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

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

TABLE 4.1: Hardware configuration of the control machines.

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

4.3.1 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.
- The final target user is the most professional-oriented one, involving users like "staking companies" or "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.

The hardware resources chosen to test the clients are summarized in table 4.1. 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 inside section 4.4, where the analysis of the results is extended.

4.3.2 Client Configurations

Ethereum has considerably emphasised on client diversity as a measure to increase the network's resilience to human-generated bugs in code. 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. Each client can easily exceed the 50 distinct parameters in a complex consensus protocol like Ethereum's. 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[199] 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 checkpointing frequency, or an optimal cache size, even the simplest database schema that could fulfil most use cases. Thus, for most users, staying within the margins of these default parameters is recommended, changing only those they know they need for their particular needs.

4.3.3 Experiments Timing Setups

This chapter 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.

Synchronization on Default and Fat Nodes The chain synchronisation from genesis was performed using the available six clients on "default mode" in a "default node" and "fat-node" from table 4.1. 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

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 4.1, 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

Shynchronization of the Medalla Network The synchronization of clients in the Medalla testnet was performed between the dates displayed in table 4.2 (prior even to the official launch of the Beacon Chain). The clients were run sequentially in a "Medalla node" from table 4.1 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 4.2: 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

4.3.4 Tooling

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

Node Exporter

Each control machine we have used to perform the study has been entirely monitored by the software tool *Node Exporter* [136], giving us access to the whole list of metrics mentioned in the previous paragraphs. Combined with a central *Prometheus* [153] 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.

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[7], EthSeer[8] or Beaconcha.in[9]. However, to support these web applications, they must interact with chain data through the REST API that most clients offer. The clients are supposed to offer a defined standard number of endpoints [58, 59]. 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 [5] that assesses any given REST API endpoint's responsiveness and robustness under varying conditions and workloads. Intending to make the fairest comparison across clients,

[7] https://etherscan.io/
[8] https://ethseer.io
[9] https://beaconcha.in/

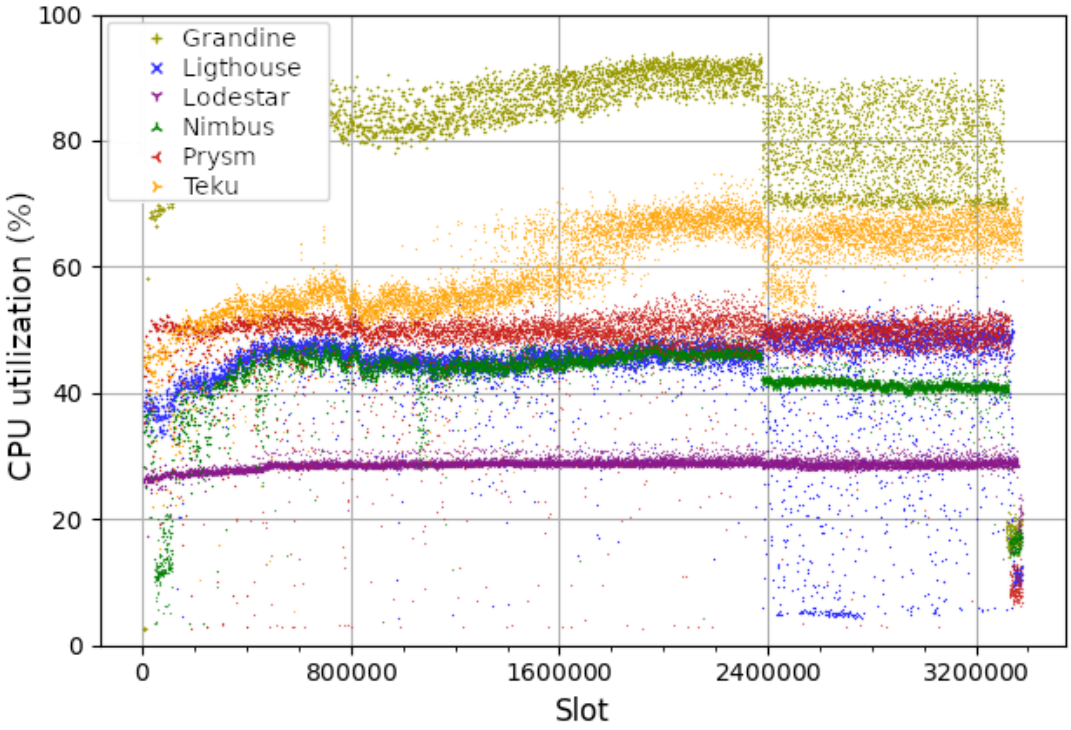

FIGURE 4.1: CPU utilization while syncing the chain from a default node. Comparison across clients.

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.

4.4 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.

The following section discusses our findings, reviewing the most critical points 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 [39].

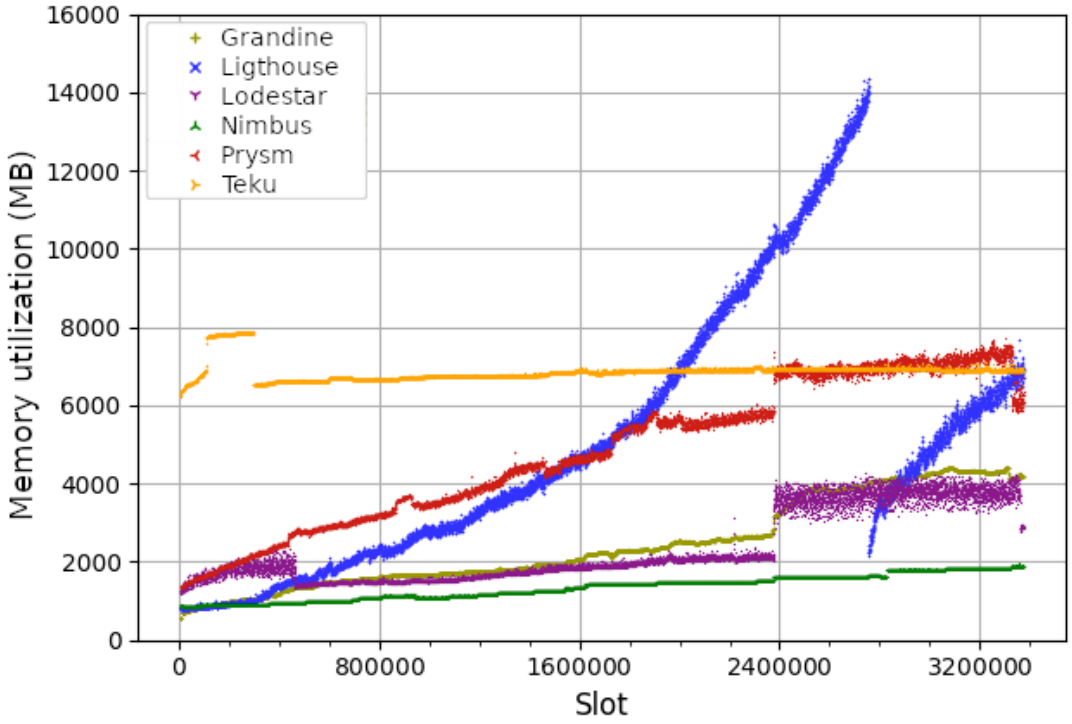

FIGURE 4.2: Memory utilization while syncing the chain from a default node. Comparison across clients.

4.4.1 Synchronization process

The entire chain synchronisation implies 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.

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 4.3.3. 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.

CPU Utilization

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 4.1 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.

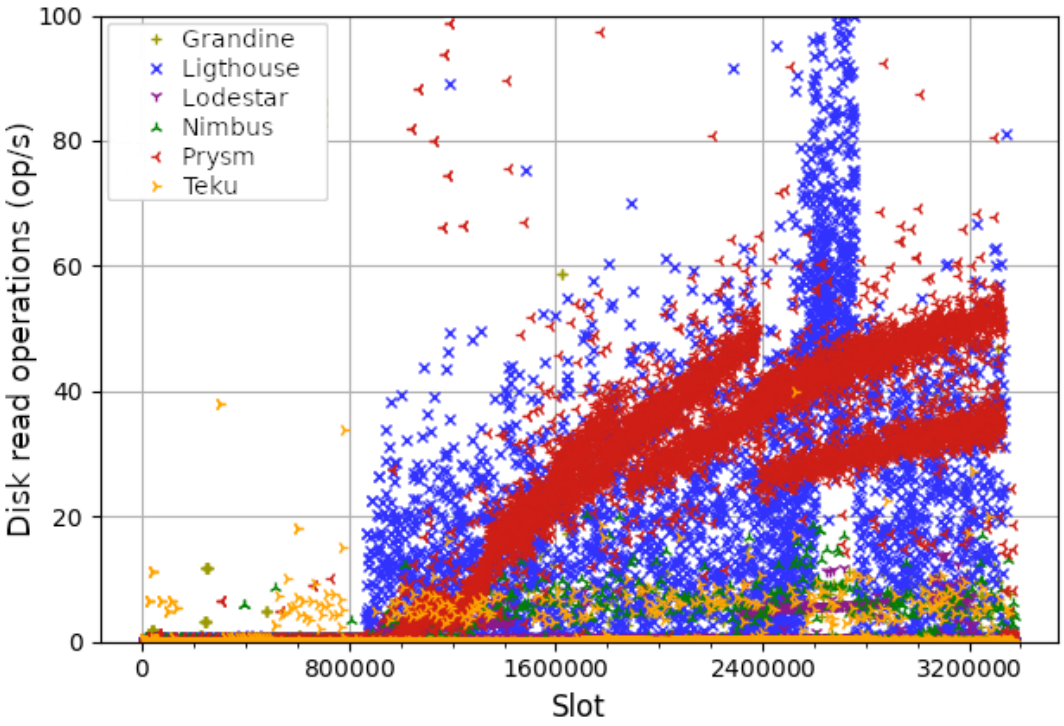

FIGURE 4.3: Disk read operations/s while syncing the chain from a default node.

Memory Utilization

In terms of memory, all the clients were exposed to different behaviours. Figure 4.2 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[10] 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,

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

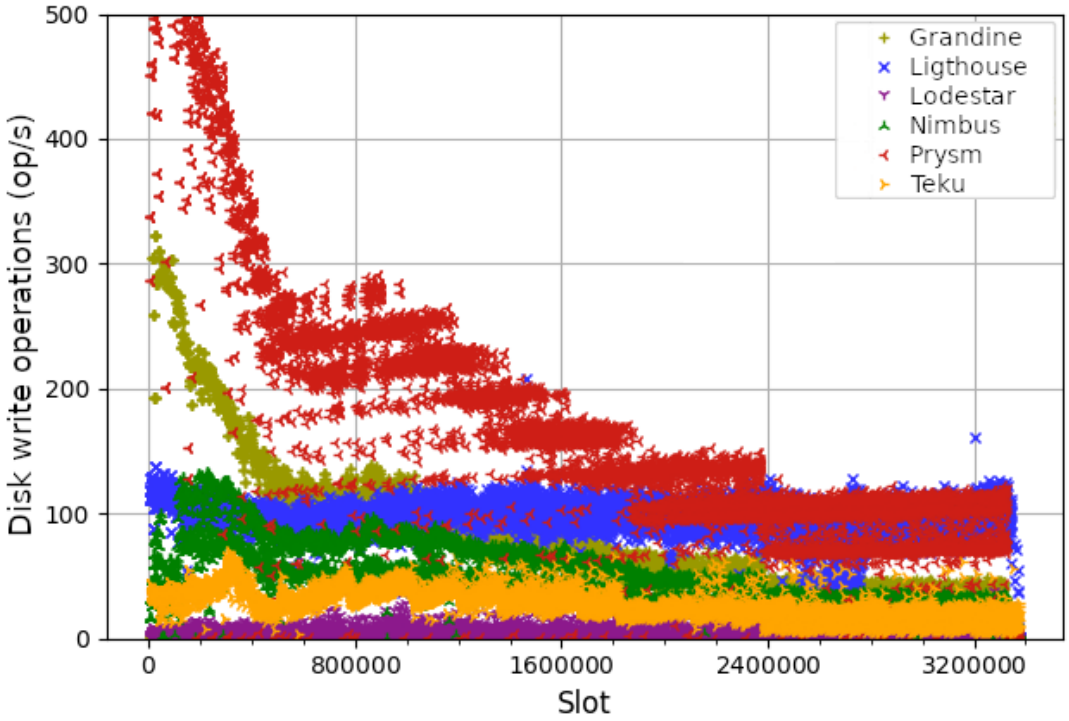

FIGURE 4.4: Disk write operations/s while syncing the chain from a default node.

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[11] (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 it 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 $7GB$. 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 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 $2GB$ 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

[11] https://sigmaprime.io/

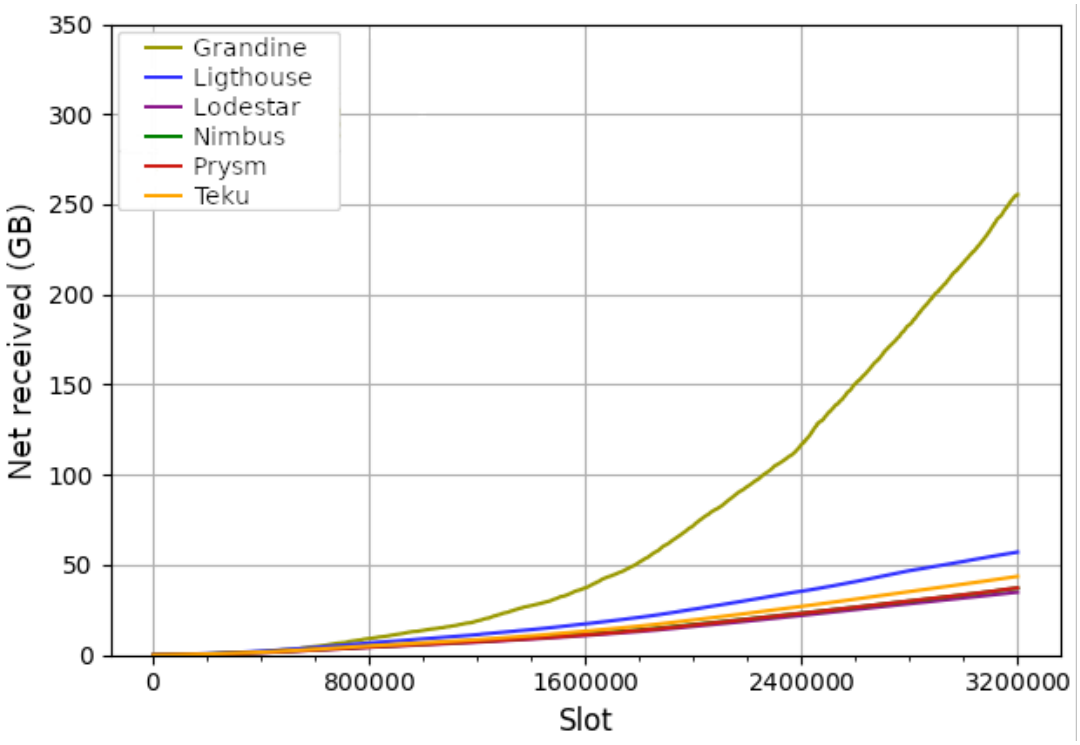

FIGURE 4.5: Network inwards utilization while syncing the chain from a default node.

bytes in memory.

Disk utilization

The differences are pretty impressive when comparing the distinct size it took to keep the entire chain by the clients. Table 4.3 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 4.3 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 in all the clients, and it might be important to tune up based on the necessities and resources of each user.

The disk utilization has some other peculiarities, though. With a wider focus on disk write operations in figure 4.4, most clients, and in particular Prysm, have

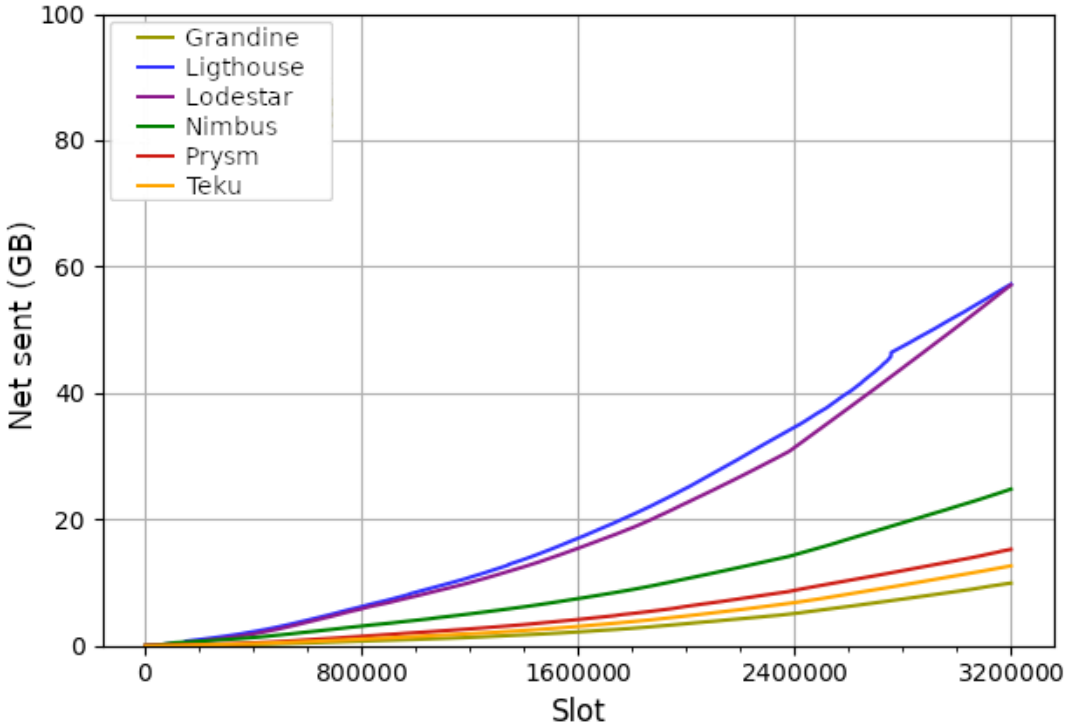

FIGURE 4.6: Network outwards utilization while syncing the chain from a default node.

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

TABLE 4.3: Total disk usage of the beacon node's database keeping beacon blocks and states after syncing the chain on the default mode.

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

TABLE 4.4: Target of peer connections for each client during the chain synchronization on a default configuration.

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 4.4 and 4.3 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 4.3 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 $900K$, 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.

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 4.5 and 4.6 show the aggregated received and sent *GB* throughout the synchronization of the chain.

Figure 4.5 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 the same blocks multiple times.

Regarding sent GBs, figure 4.6 shows that only Lodestar and Lighthouse are above the rest of the clients with around five times more output bandwidth than the "quieter" clients. As mentioned, bandwidth usage highly depends on each client's simultaneous peers. Table 4.4 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.

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 "performance" as 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 4.7 shows each client's total duration of the synchronization process. The figure shows that concurrency has clear benefits in speeding up 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.

4.4.2 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.

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 at which the chain checkpoints are stored in the database, making all the RPC queries generally faster as it takes less time to recompose intermediary states. Making it ideal for those users who want to have faster access to chain data. The whole concept of checkpoint syncing relies on the node's 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

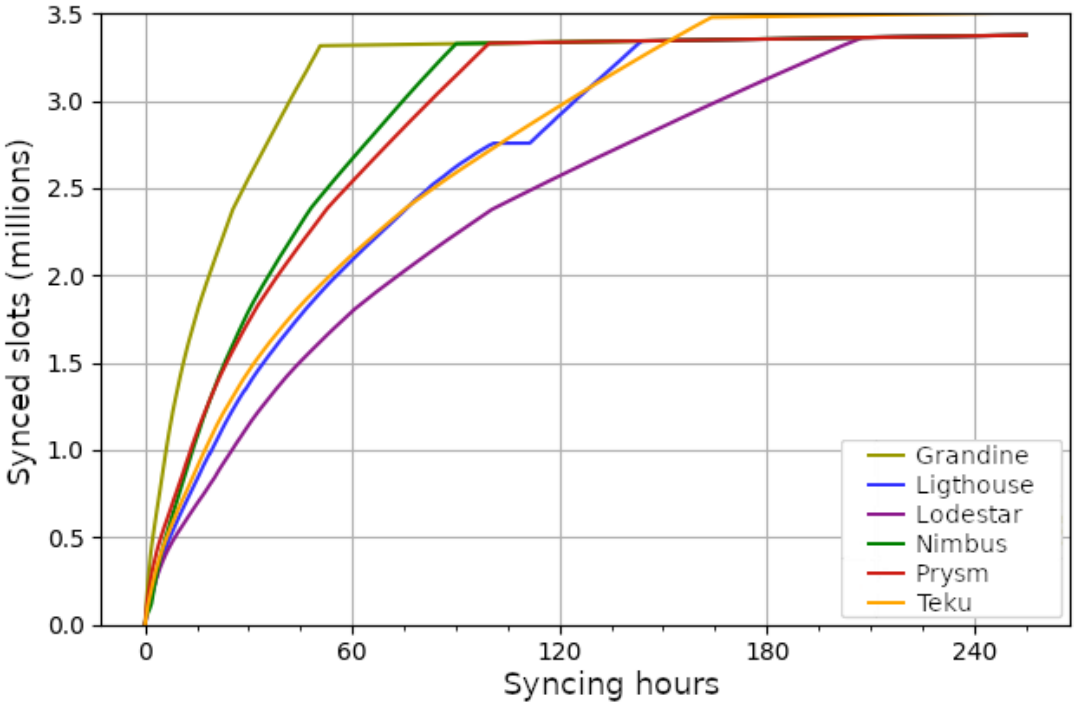

FIGURE 4.7: Chain synchronization speed as the number of slots downloaded and processed since the node on standard hardware was run.

a trusted node, allowing it to process and validate incoming new blocks after the process has been accomplished[12]. 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 4.8 shows that clients achieve a better synchronization speed with more capable hardware. On average, our measurements show a speed improvement of 135.2%, with Lodestar showing a disappointing performance decrease of 10% and Grandine syncing a remarkable 178.5% faster. All the measured slot processing ratios per second are displayed in figure 4.9, where we can appreciate the average performance of Teku, Nimbus, Prysm, and Lighthouse, and the outlying one from Grandine and Lodestar.

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 section 4.4.4, 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 4.10 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

[12]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 chapter, 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.

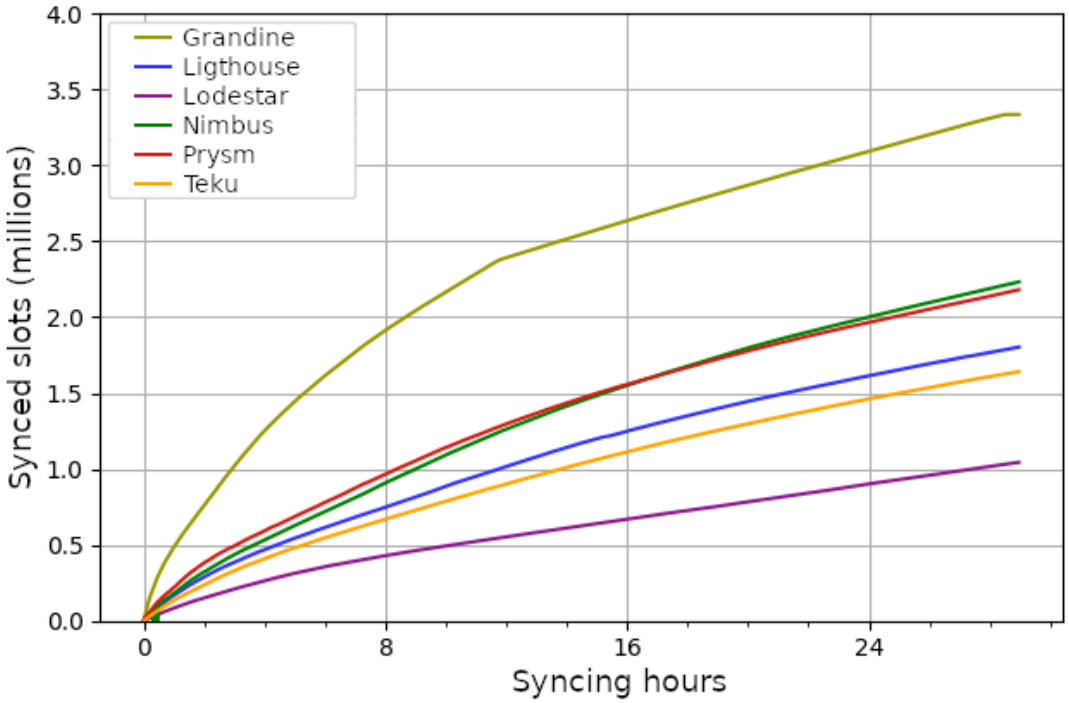

FIGURE 4.8: Chain synchronization speed as number of slots downloaded and processed since the fat node was run.

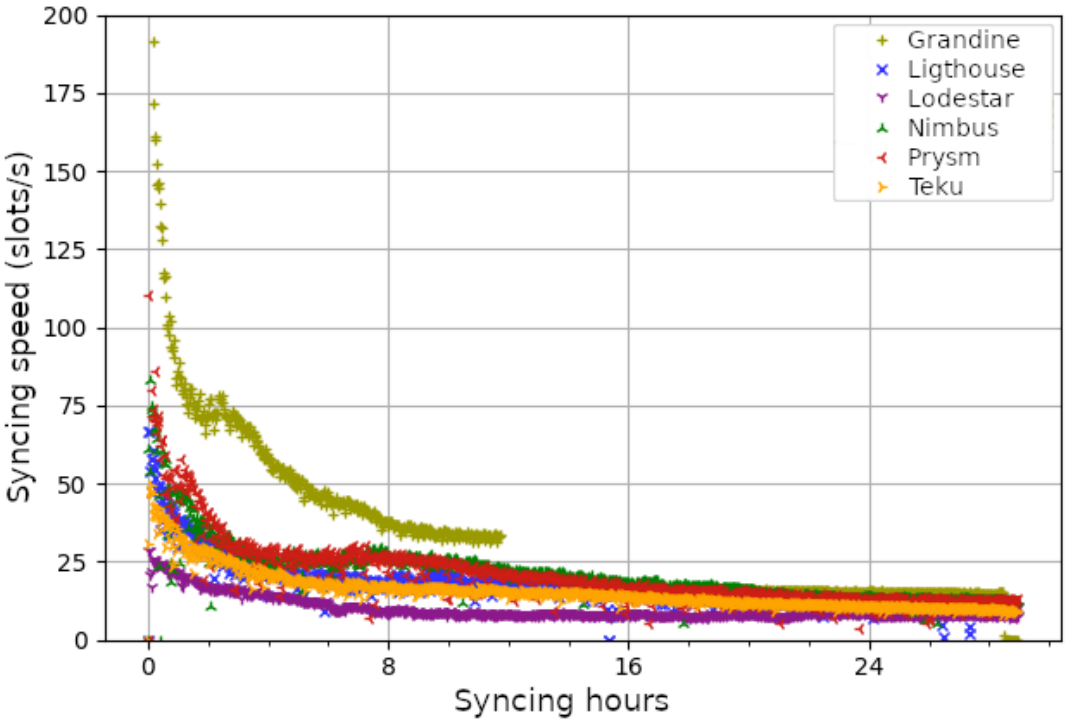

FIGURE 4.9: Slot synchronization ratio per second for the different clients.

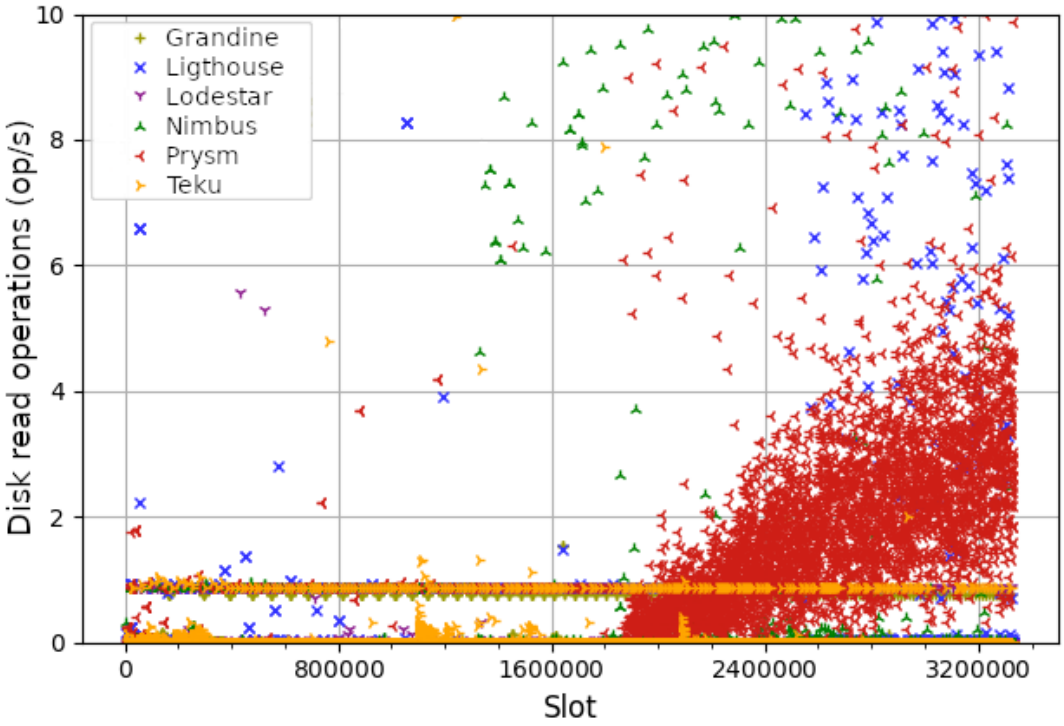

FIGURE 4.10: Disk read operations by the clients while running in a "Fat" node.

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 4.11 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.

4.4.3 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 impossible 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 the synchronization tedious because of the state processing duration. For timing reasons, we've extended the section to explore the performance of the synchronization process in the Kiln[13] testnet.

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

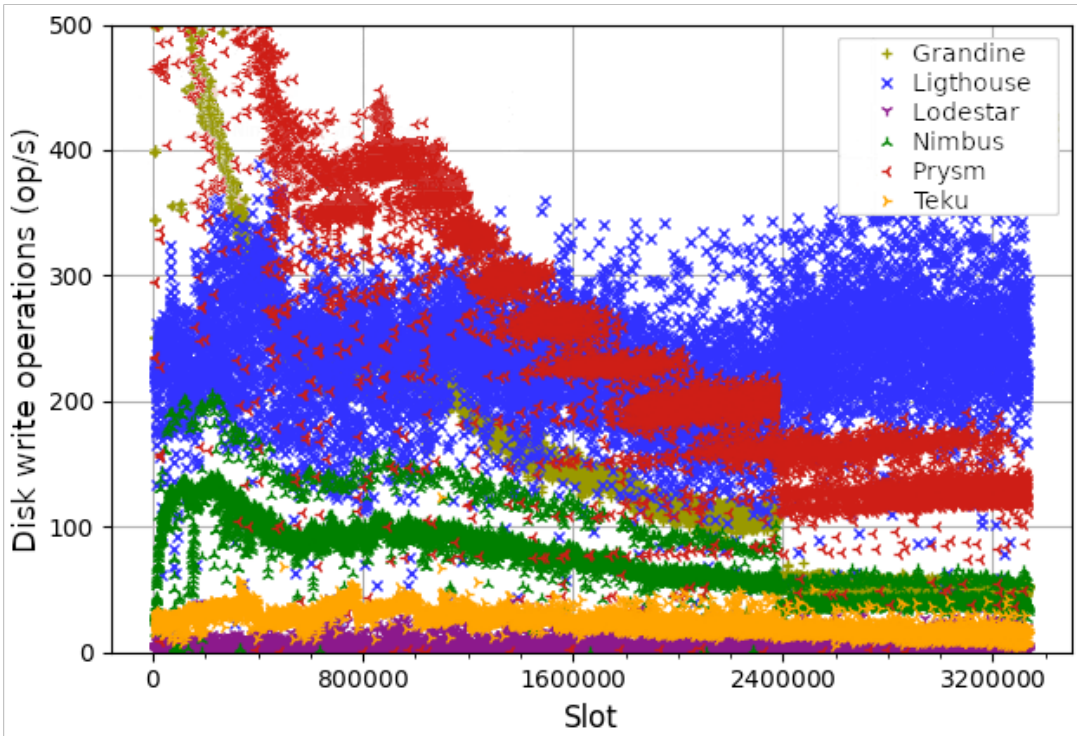

FIGURE 4.11: Disk write operations by the clients while running in a "Fat" node.

Considering that the beacon state is append-only, we expect that the required hardware resources will just 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 the available resources must be duplicated to run both system parts together.

Although there is a tendency to require faster and more extensive resources, we see 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. We suggest opting for a still low-profiled and low-powered device such as an Intel NUC or similar.

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 table 6.1, 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 on the same range of dates (18th of March, 2022).

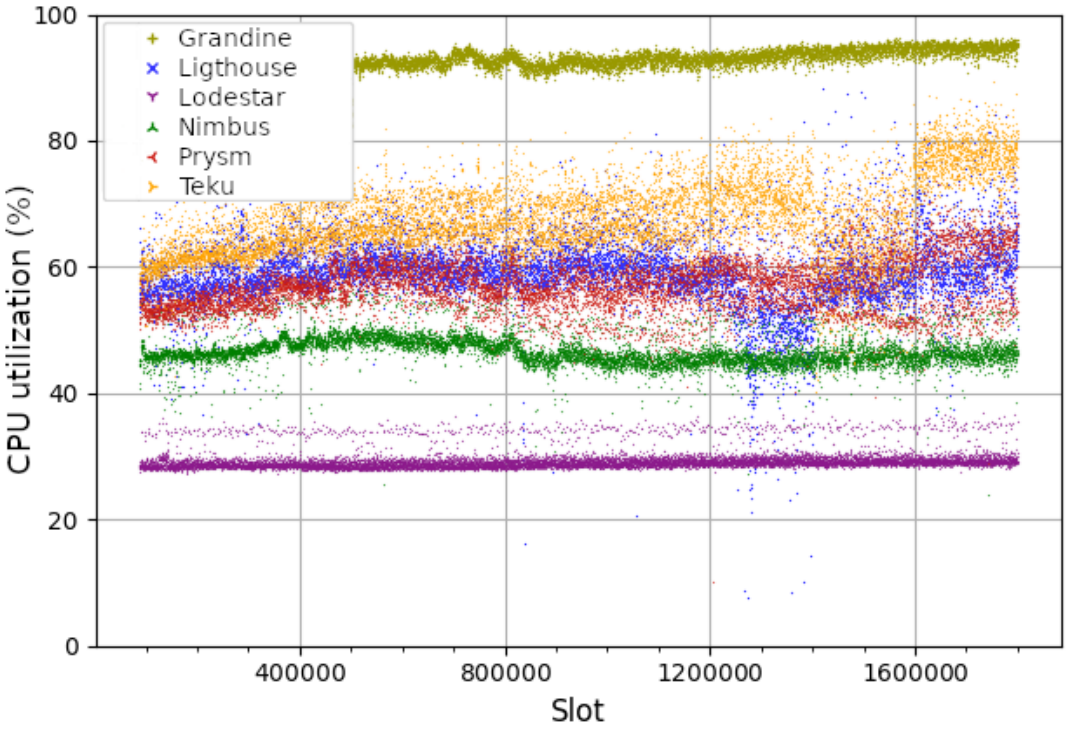

FIGURE 4.12: CPU utilization while syncing the chain in a Raspberry Pi 4.

CPU Utilization

Figure 4.12 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.

Memory Utilization

A similar pattern was observed in terms of memory. Figure 4.13 shows the memory allocation patterns from the CL clients in the Raspberry Pis. Teku keeps a steady performance by assigning $6GB$ of memory to the JVM. Prysm keeps allocating memory as *Go* does not free memory unless the OS asks for it, reaching $5GB$ of memory in its latest synchronization stages. Lodestar, Grandine, and Nimbus remain with the lower profile at a lower limit of $3GB$, 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 $8GB$. We experienced three client crashes as more memory than the available was asked. In this sense, each sudden drop in figure 4.13 belongs to each of the restarts of the client after the crash. As figure 4.14 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

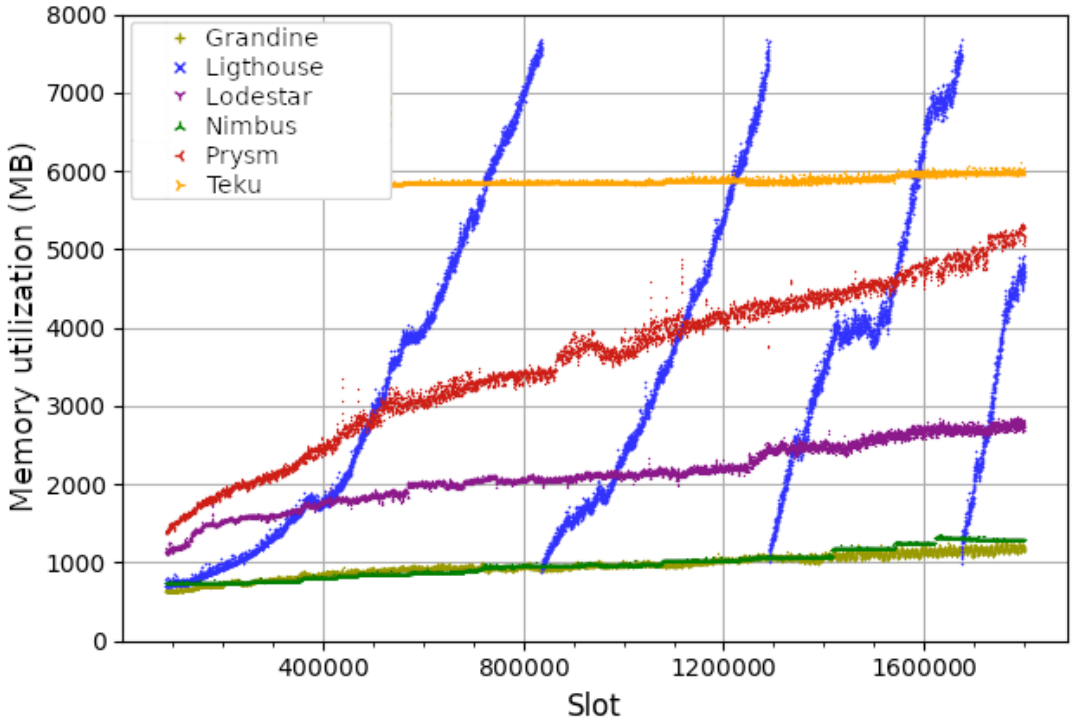

FIGURE 4.13: Memory utilization while syncing the chain in a Raspberry Pi 4.

leak, Lighthouse highly relies on disk write operations compared to other clients, multiplying by three times the usage of Teku and by more than eight times the rest.

Performance

In the opposite direction as the one measured with the "fat nodes", figure 4.15 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 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.

4.4.4 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 4.3.3.

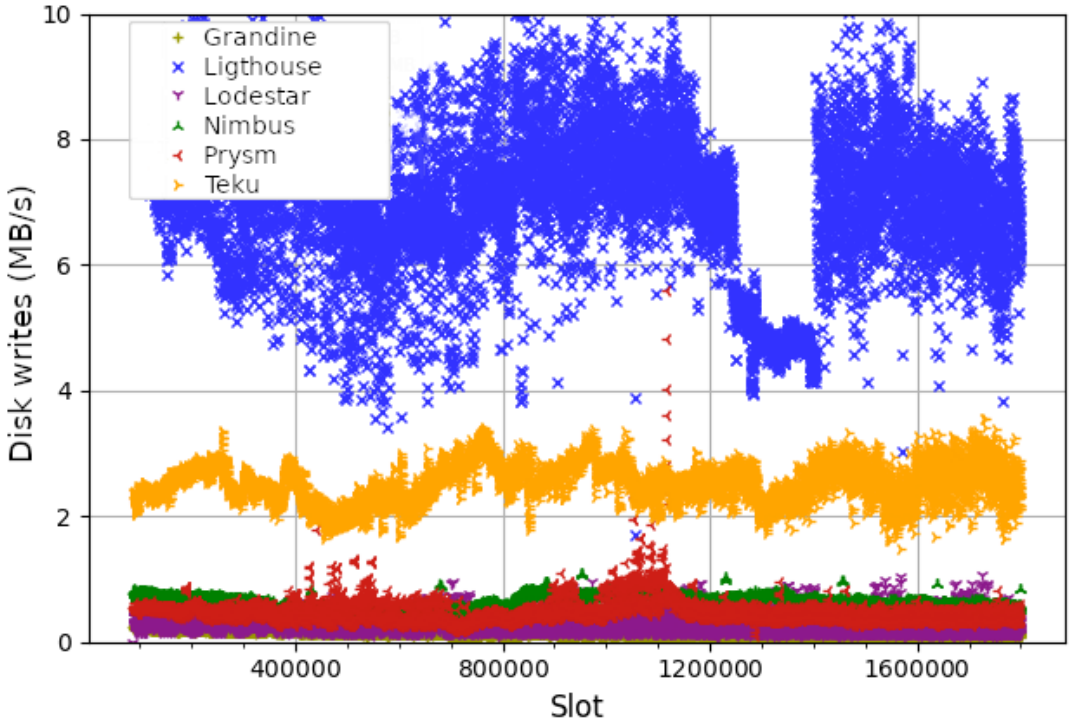

FIGURE 4.14: Disk writing speeds while syncing the chain in a Raspberry Pi 4.

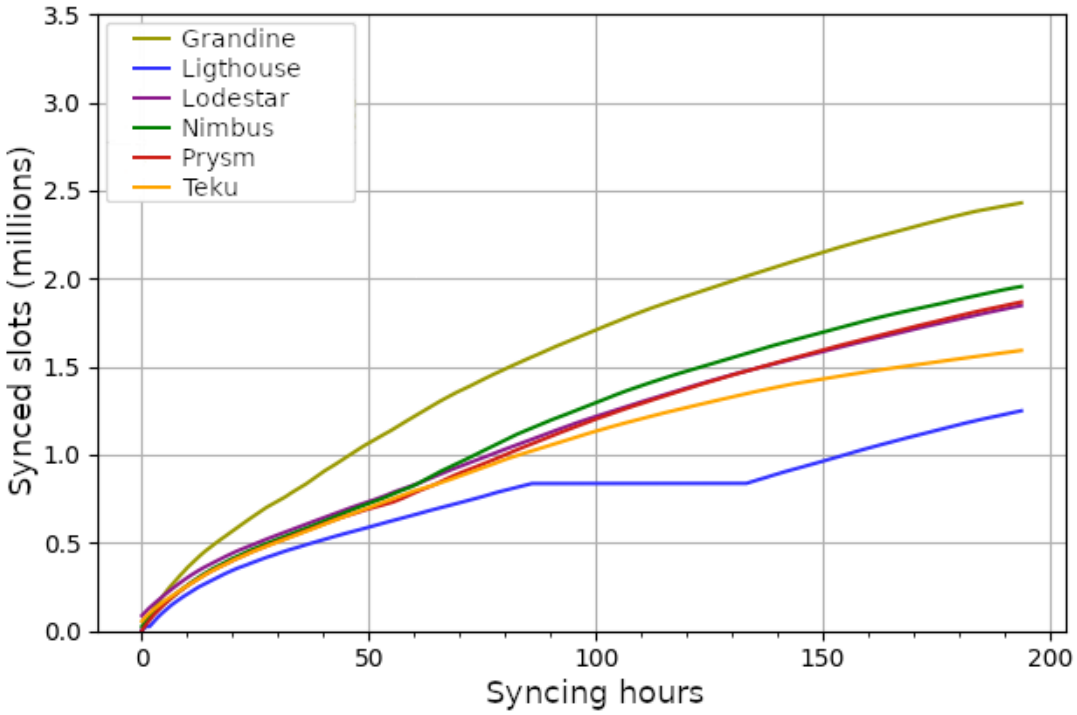

FIGURE 4.15: Chain synchronization speed as the number of slots downloaded and processed since the node on the Raspberry Pi 4 was run.

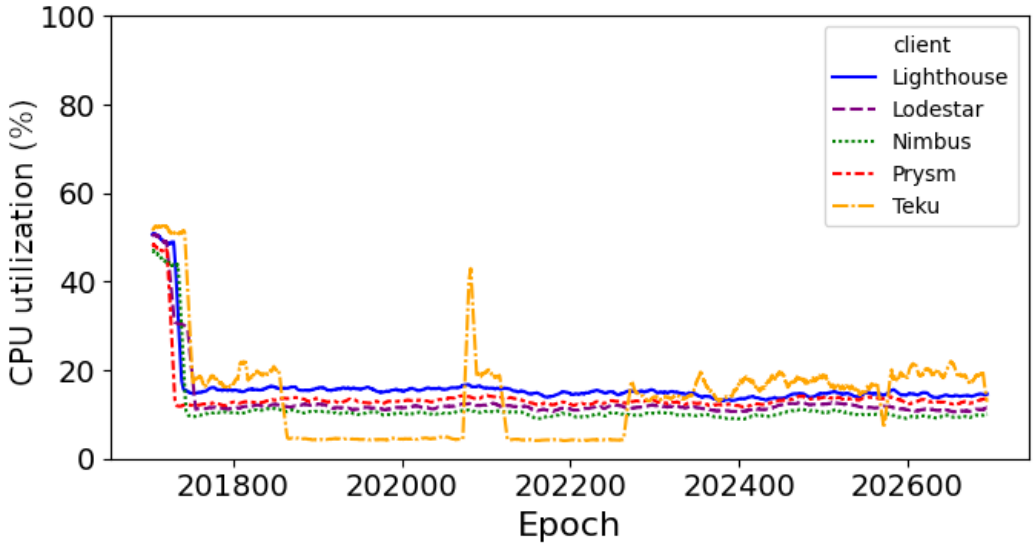

FIGURE 4.16: CPU usage by clients following the head of the chain.

CPU Utilization

Unlike the chain synchronization process, the regular operation of EL and CL clients doesn't require much CPU. Figure 4.16 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.

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.

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 4.17 shows that the client difference is more noticeable than the

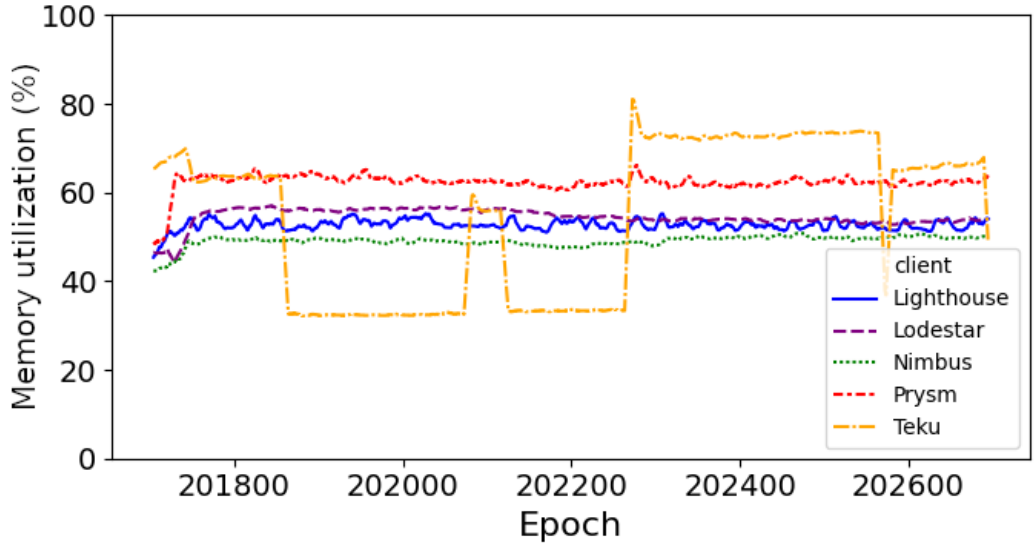

FIGURE 4.17: Memory usage by clients following the head of the chain.

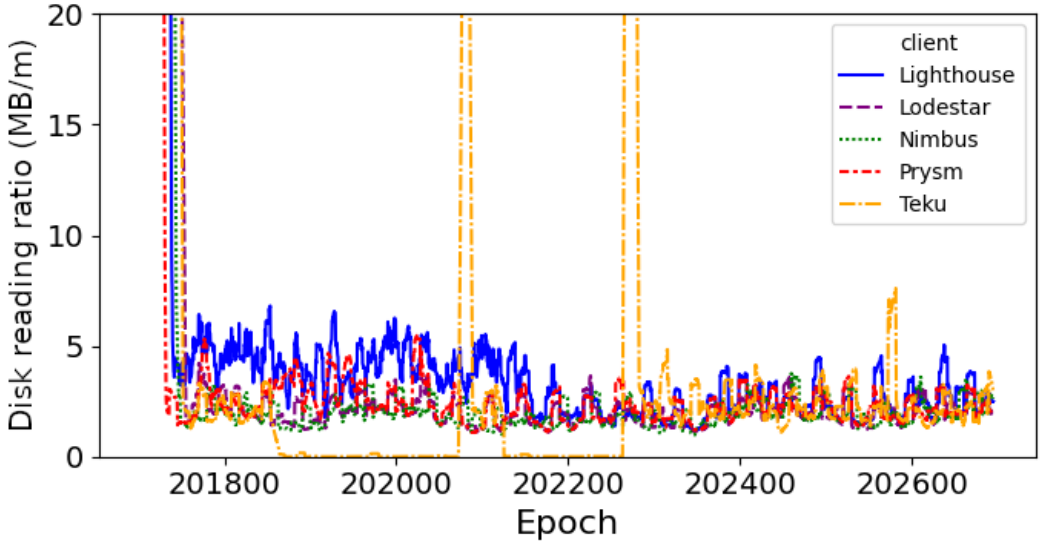

FIGURE 4.18: Disk reads by clients following the head of the chain.

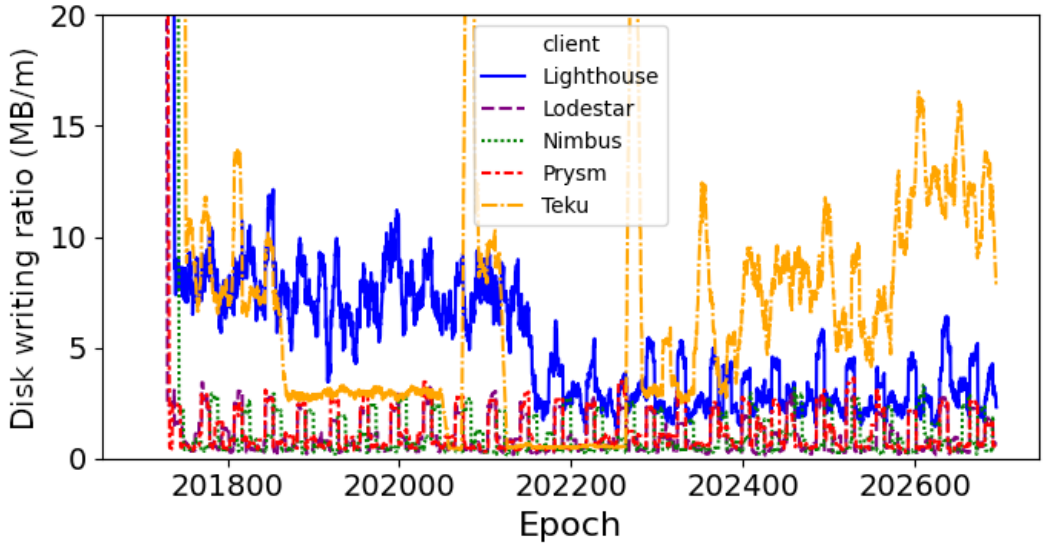

FIGURE 4.19: Disk writes by clients following the head of the chain.

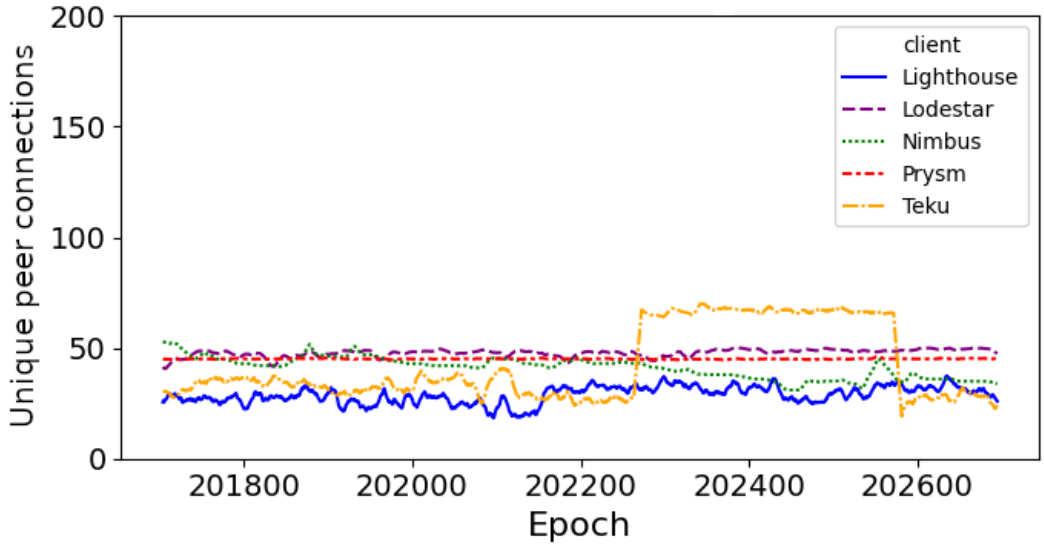

FIGURE 4.20: Concurrent node connections by the clients while following the head of the chain.

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.

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.

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 4.4.1, where clients could handle the persisting of blocks and states to DB in a more customized or optimized way, neither figure 4.18 nor figure 4.19 show clear differences in the number of bytes read or written into the disk. Figure 4.18 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 4.19 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.

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.

Table 4.5 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 stay fluctuate more between the 30 to 40 concurrent connections. Comparing these values to the previous ones presented in table 4.4, 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.

To observe the impact of these concurrent connections in the downlink, we can use figure 4.21, 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

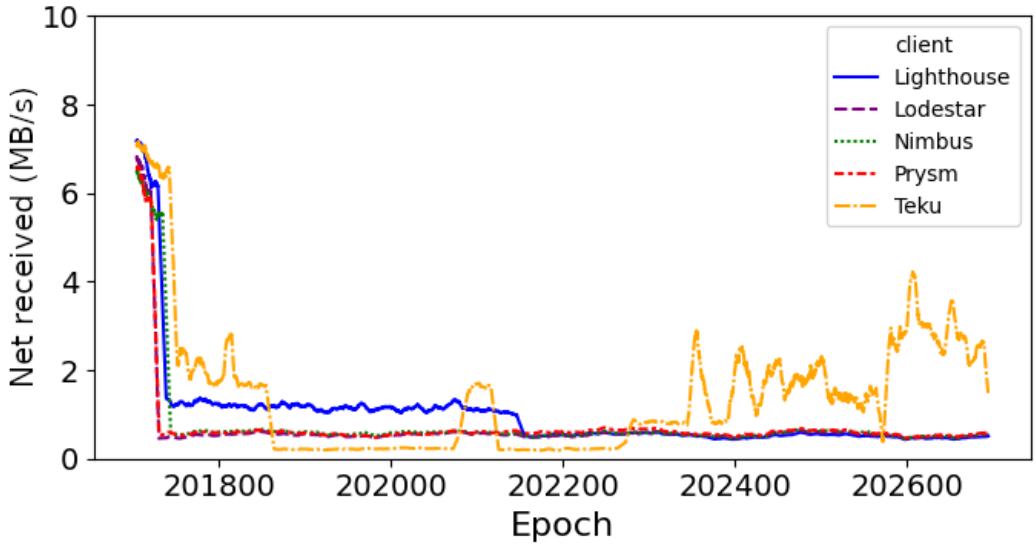

FIGURE 4.21: Network incoming bandwidth for the clients while following the head of the chain.

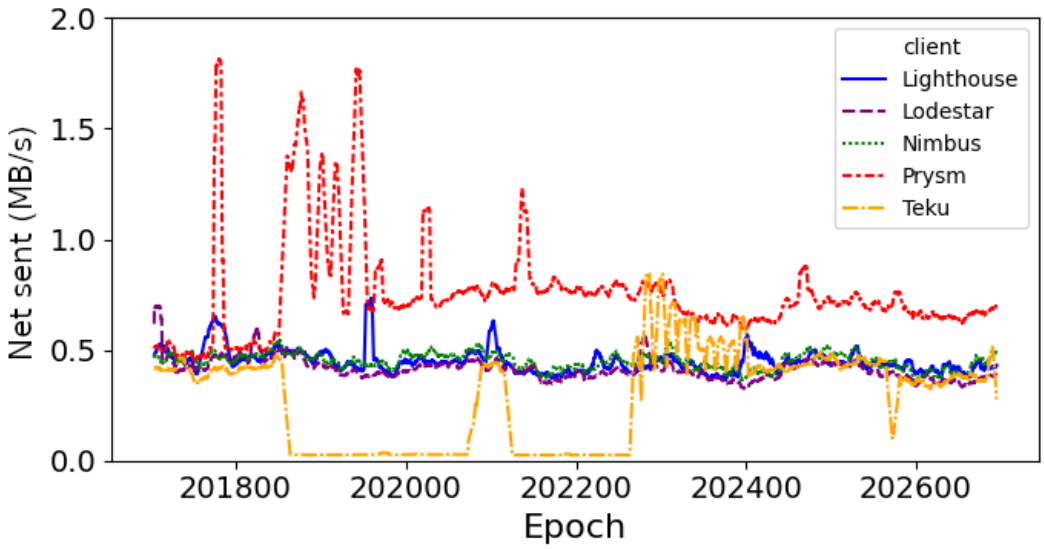

FIGURE 4.22: Network incoming bandwidth for the clients while following the head of the chain.

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

TABLE 4.5: Target of peer connections for each client during its regular operation.

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 4.22 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.

4.5 Hardware and network observations

The wide variety of data presented in the previous section 4.4 compiles 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 chapter, we analyse how the Ethereum protocol, network instabilities, or the user's interaction impact the resource utilization of the different clients.

4.5.1 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 [25] defines the propagation and arrival of these duties as asynchronous, the rewards clearly incentive validators 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 section 2.3.2. In the section, and more in detail while looking at figure 2.5, we can appreciate how each duty is followed by a 4 second window to propagate each message over the network.

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 4.23 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 aggregations on it, and update the existing beacon state with the new block. It

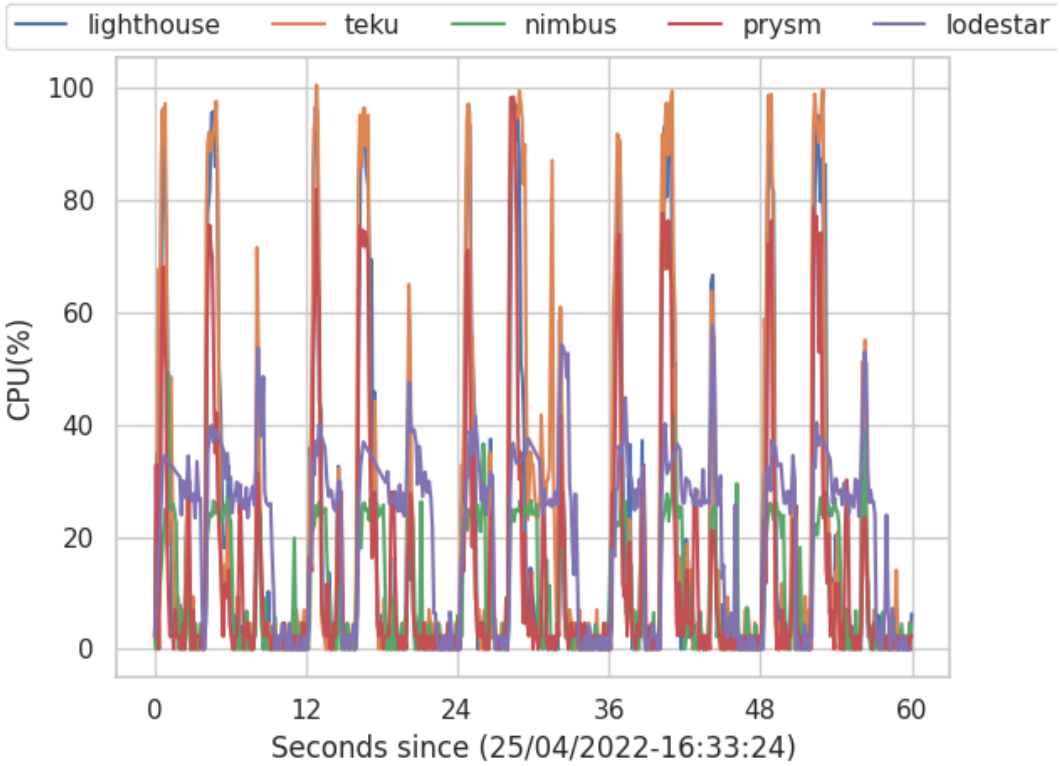

FIGURE 4.23: CPU utilization at the slot range while following the head of the chain.

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

	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

TABLE 4.6: 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.

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.

4.5.2 Hardware Resources Evolution over Hardforks

Setting up a requirements list to participate as a full Ethereum node is difficult. Previous sections 4.4.1 4.4.2 and 4.4.3 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 4.6 and 4.7 show the mean resource utilization of each client on the different stages and hardforks of the chain. Table 4.6 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.

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 4.5.1. This heavily reduces the CPU utilization. Table 4.7 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

	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

TABLE 4.7: 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.

to previous hard forks, the resources taken by the clients to run after Capella are considerably higher:

- the CPU utilization has reduced between 59.90% and 91.76% of the syncing values (mean decrease of 73.94%).
- the memory usage has increased between 153.42% and 939.98% (mean increase of 425.52%).
- the incoming network bandwidth utilization has increased between 478.74% and 1339.53% (mean increase of 824.15%).
- the outgoing network bandwidth utilization has increased between 838% and 9975% (mean increase of 4320%).

We must note that after “the merge", the execution and consensus clients needed to be paired individually. Thus, the measurements in table 4.7 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)[14] to help increase the resilience of the network or further protocol upgrades like ProtoDankSharding and DankSharding[15] 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.

4.5.3 Perceiving Network Instabilities from Chain Synchronization

Back to the first stages of this study, when we were putting to test our methodology with the Medalla testnet (as described in section 4.3.3), we identified that syncing up the historical records of the chains could give more information than one would imagine.

At the moment of syncing the testnet, the slot synchronization speed shown in figure 4.24 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.

[14]https://ethereum.org/en/staking/dvt
[15]https://ethereum.org/en/roadmap/danksharding

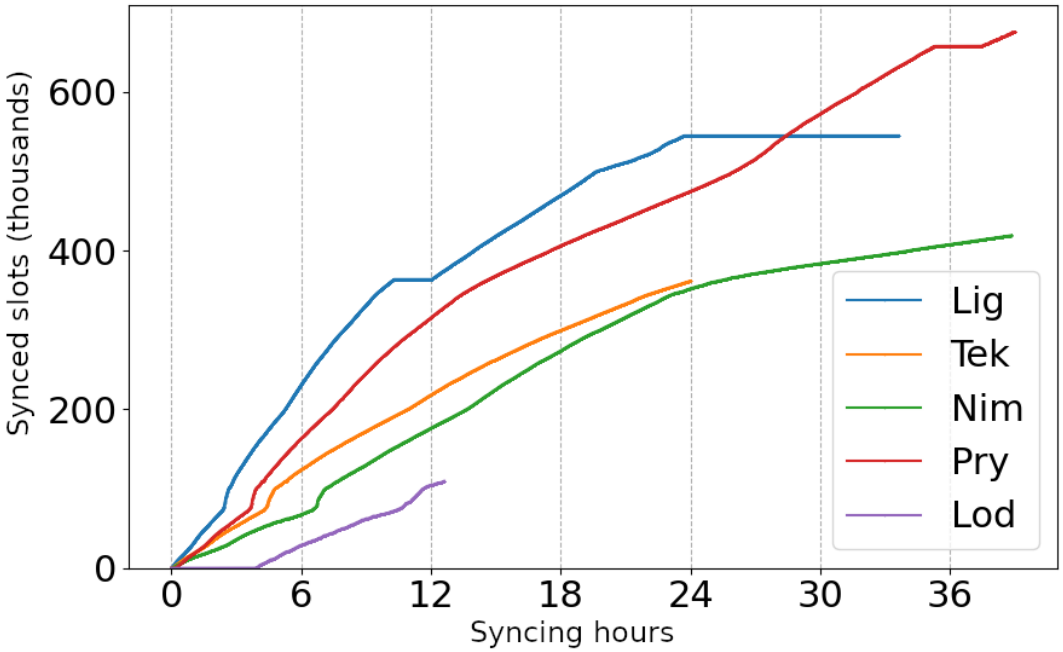

FIGURE 4.24: Chain synchronization time by clients in the Medalla testnet.

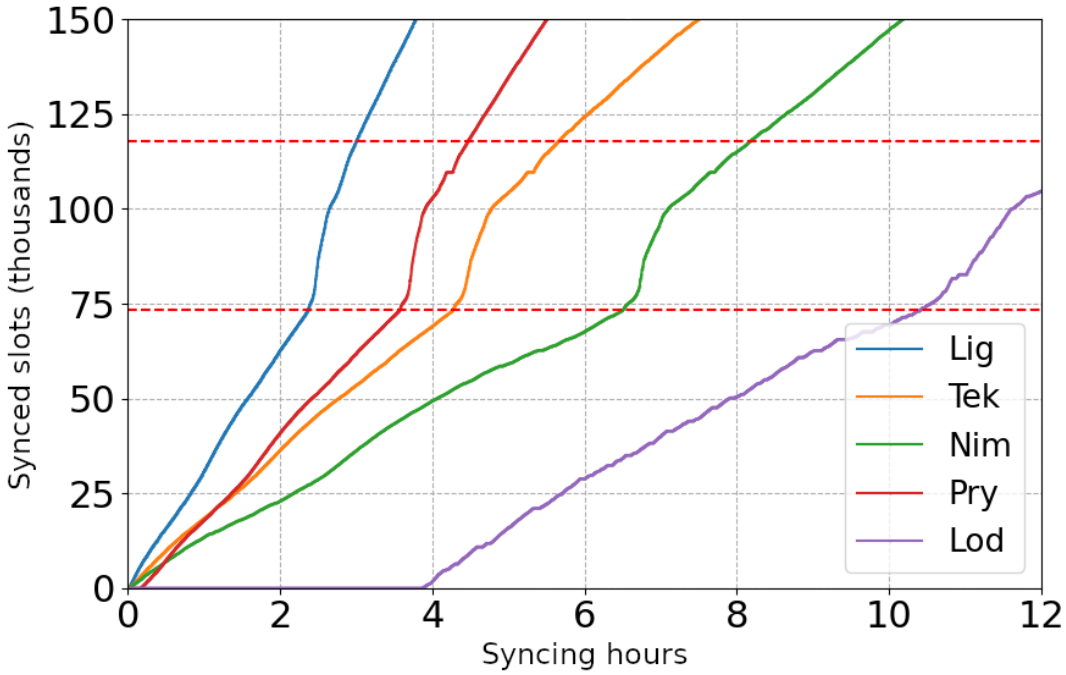

FIGURE 4.25: Zoomed slot synchronization speed of consensus clients in the Medalla testnet.

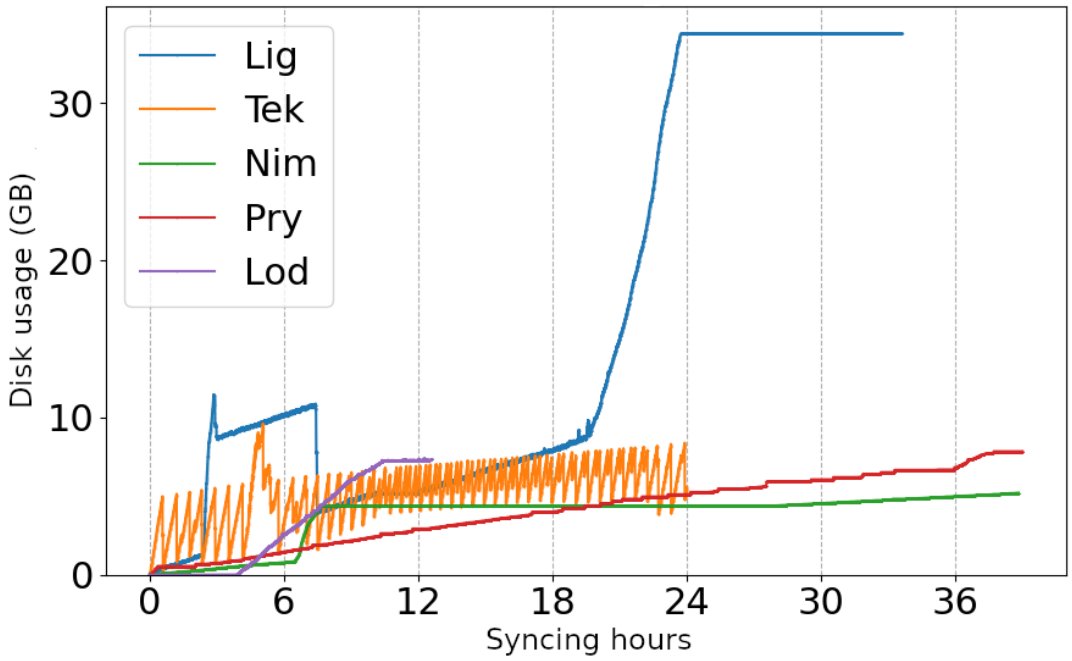

FIGURE 4.26: Disk usage of consensus clients during the Medalla testnet synchronization.

But 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 4.25 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 4.26 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 both anomalies, figure 4.27 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 [124].
- Figure 4.27 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.

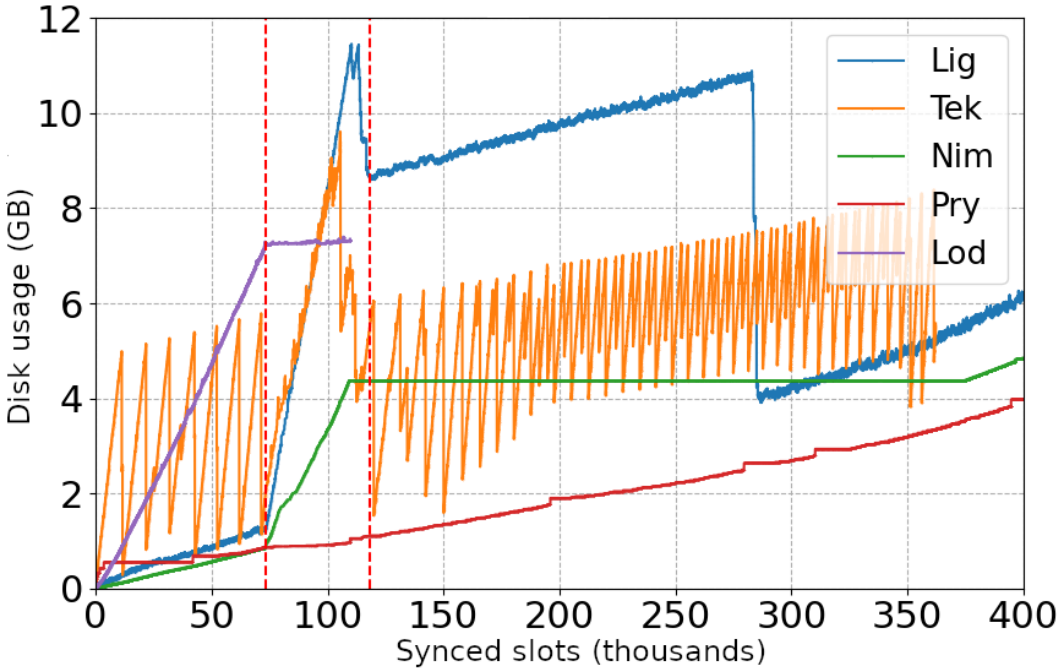

FIGURE 4.27: Disk Usage of Ethereum clients while syncing the Medalla testnet using the slots as X axis.

On the other hand, Nimbus's rise in disk storage is not as sharp as Teku and Lighthouse. However, it did not reduce its storage 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 4.28, 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, there were blocks 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

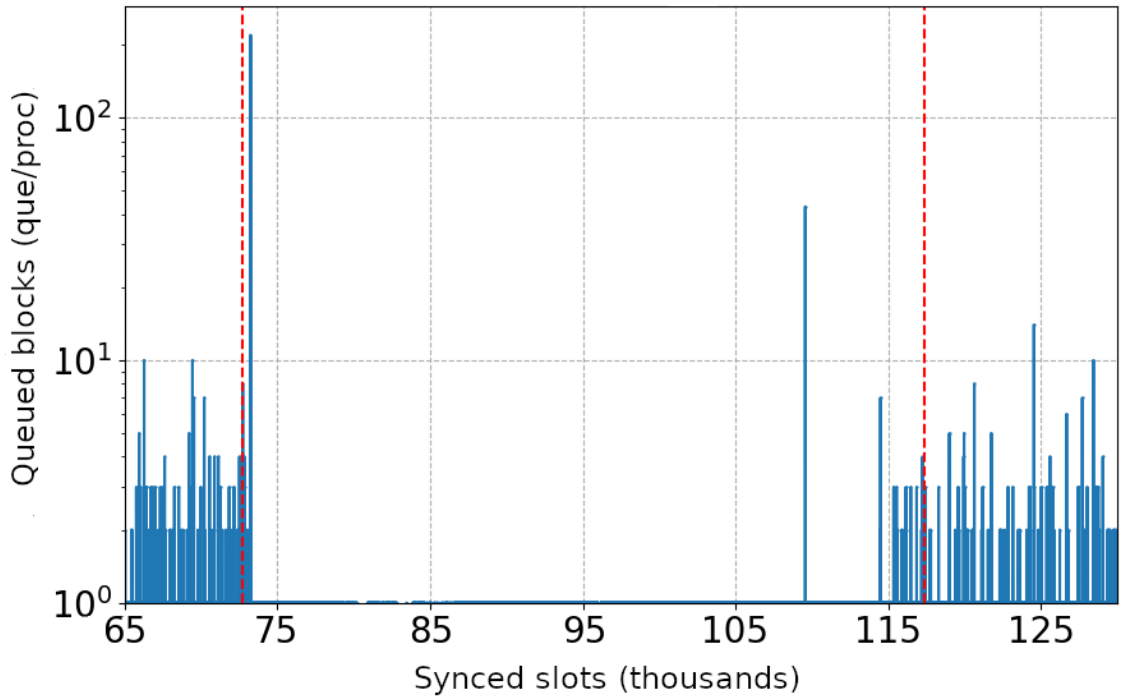

FIGURE 4.28: Number of times a block is queued to be persisted while syncing the Medalla chain on Lighthouse. Zoomed at the non-finality period.

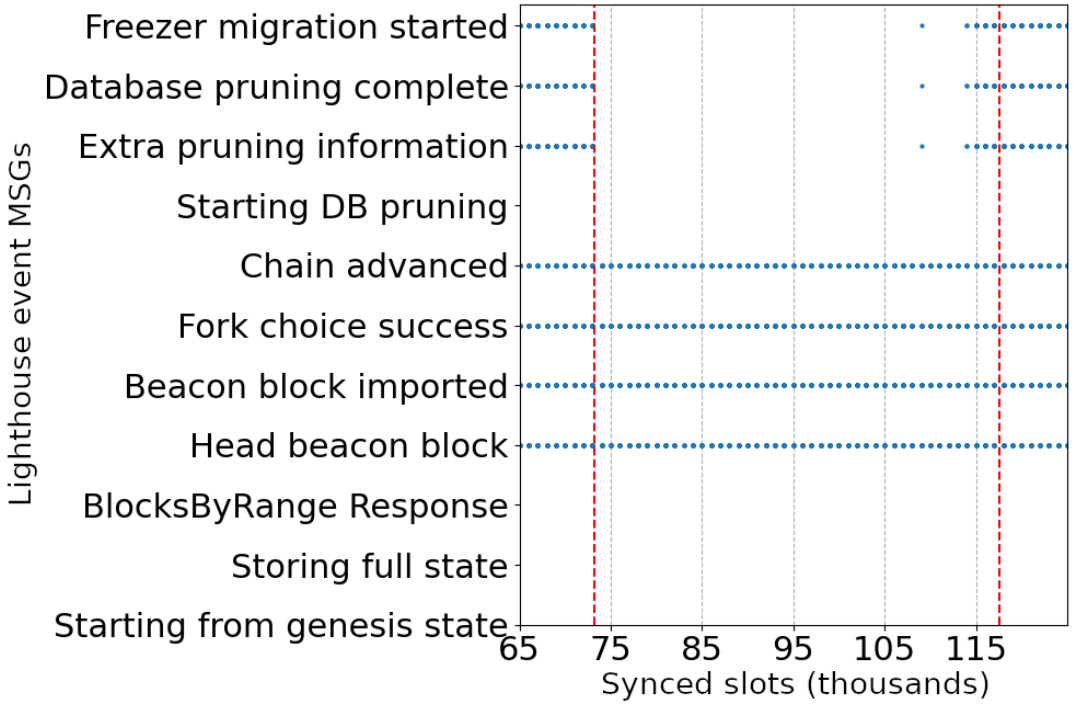

FIGURE 4.29: Events timeline for Lighthouse while syncing the Medalla testnet. Zoomed at the non-finality period.

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

TABLE 4.8: Percentage of responses the Beacon node could successfully reply under different concurrent request workloads.

states during a non-finality period. Thus, to corroborate our hypothesis, we analyzed Lighthouse's detailed logs and plotted the frequency at which different events get executed. Figure 4.29 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, which is consistent with the rapid disk usage increase.

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

4.5.4 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 4.3.4. 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 4.8 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 4.30, 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.

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 4.8 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 4.31, 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

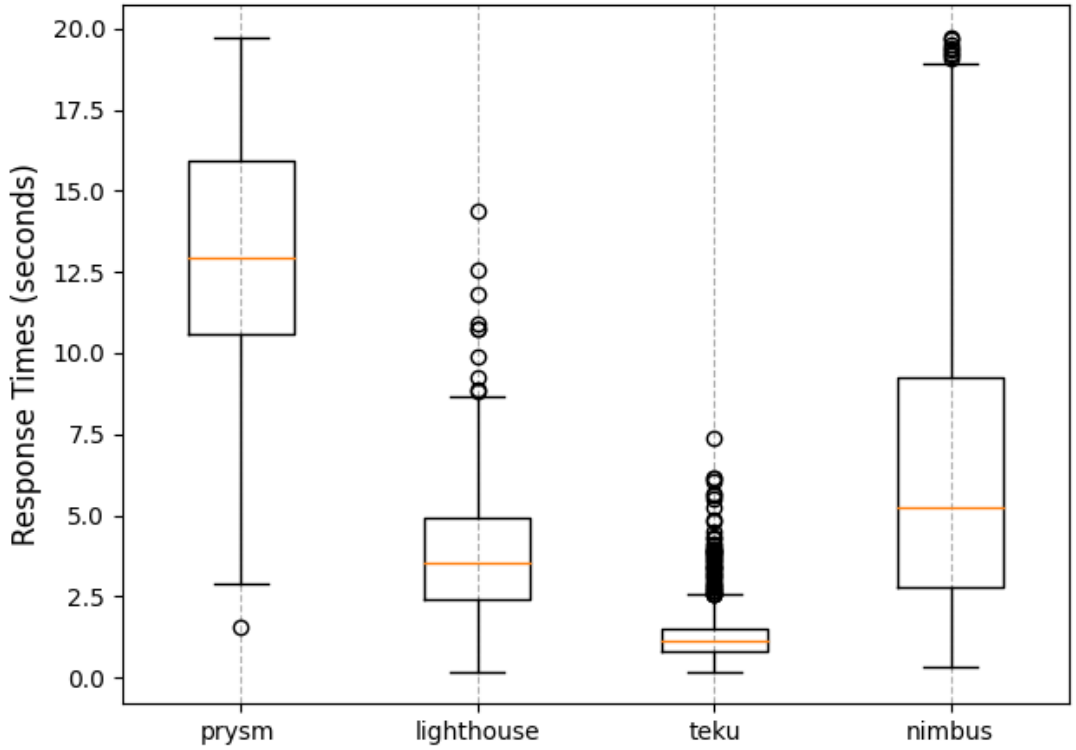

FIGURE 4.30: Distribution of the Beacon node's response time under a single concurrent request workload. Note that 20 seconds was the timeout of the request.

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[16].

4.6 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.

[16]https://www.infura.io/

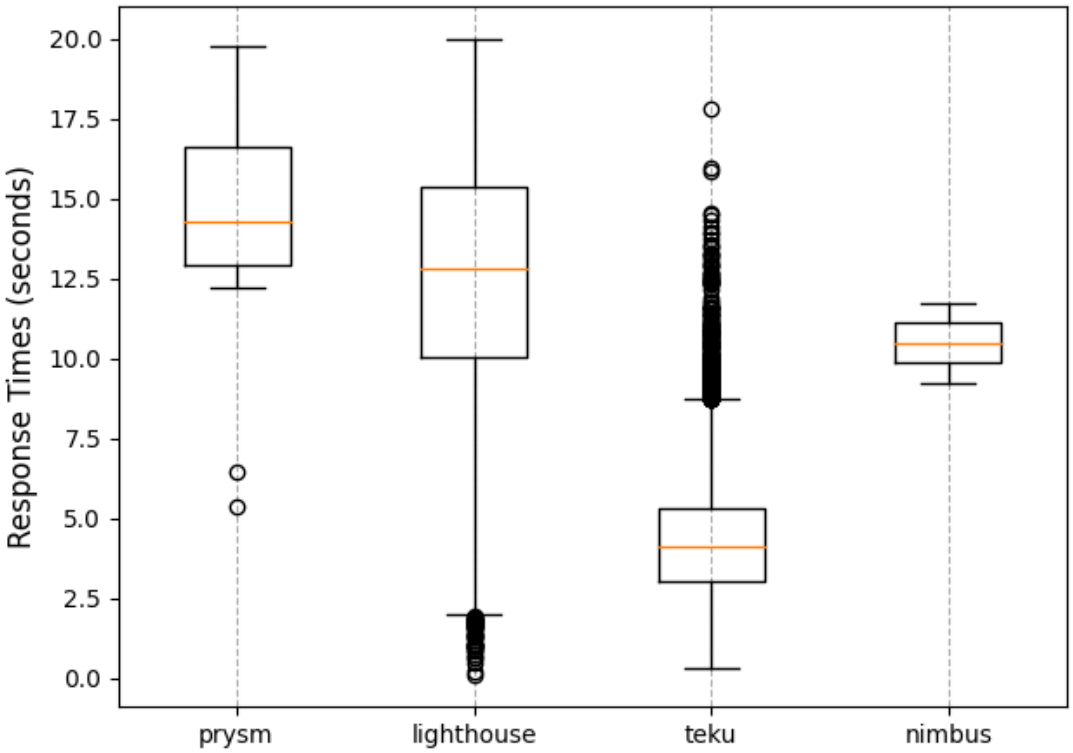

FIGURE 4.31: Distribution of the Beacon node's response time under ten concurrent requests workload. Note that 20 seconds was the time-out of the request.

4.6.1 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.

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[17] 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

[17]https://prysmaticlabs.com/

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.

Lighthouse from Sigma Prime[18] 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.

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[19] uses it to provide all the information through Infura[20]. 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.

Nimbus from Status[21] 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.

Lodestar from ChainSafe[22] 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.

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 being still many features in beta, clearly, the biggest drawback of this client is that it has not been open-sourced yet.

[18] https://sigmaprime.io/
[19] https://consensys.io/
[20] https://www.infura.io/
[21] https://status.im/
[22] https://chainsafe.io/

4.6.2 Hardware Recommendations

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 believe that the client's current requirements and performance exceed the hardware of domestic low-performance hardware devices such as 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 chapter, 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.

4.6.3 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.

4.7 Conclusion

In this chapter, we have shown multiple aspects of all Ethereum CL clients while tested under different conditions. We have exposed their strengths as well as 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 they 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 the strong software diversity that Ethereum has, and this is hard to find anywhere else in the blockchain ecosystem. Overall, 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.

Chapter 5

Incentive Changes with Ethereum's Merge

5.1 Introduction

Ethereum [208] has soundly consolidated in the Web3 and blockchain ecosystem for its widely decentralized network with over 950.000 active validators. It currently handles above 1 million daily transactions from 600 thousand active accounts [138]. However, the network hasn't been consolidated over an altruistic mindset where participants in the consensus don't get any credit for doing so. As with any other blockchain in the Web3 space, Ethereum relies on a well-studied incentive program to reward honest behaviours between the participants.

Starting as an operative network in 2015, Ethereum chose Proof of Work (PoW) as its consensus mechanism. In this PoW-powered platform, participating nodes (commonly known as miners) competed to become the next block proposer. Nodes in the network built block candidates by adding user transactions from the public "Mem-pool"[1]. Awarded with a generous tip in Ethereum's token Ether (ETH) for the first one publishing a valid block, miners would parallelly iterate over multiple nonce values[2] until the hash value of the entire block satisfies the target required by the network.

Initially, as part of the block proposal's incentives until EIP1559 [170] was applied, block proposers were also granted with the fees coming from processing the transactions included in the block plus a dynamically adjusting base reward. However, since the network transitioned to PoS, the consensus and, therefore, the finalization of the chain was directly granted to validators. These ones now have to perform a set of tasks at every epoch of the chain. Also known as duties, validators are grouped by committees, which have the major task of voting on the correctness of the proposed block (if proposed) from the slot they are assigned to. These votes are also known as attestations and are included in the beacon blocks. Validators are also grouped by sync committee, a small set of 512 validators that help sign headers of blocks to prove their validity to light clients that won't need to sync the entire chain. Based on the correctness of all these duties, validators get rewarded as an incentive to well-behave in the consensus, making honest actions in the network more economically lucrative than dishonest ones.

Both parallel chains had co-existed since December 2020, when the beacon chain was launched, until September 2022, when the chain in charge of processing transactions became consensus-dependent from the beacon chain. In this event called "The

[1] Mem-pool it Memory-pool refers to the local list of transactions that each node can keep after listening to the public transaction broadcasts.

[2] Nonce values in PoW-based blockchains is a field in the head of the block structure that miners can indistinctly modify to satisfy the hash target requirements of the chain.

Merge"[3], active validators in the consensus layer will also be the ones proposing blocks in the EVM chain, or the Execution Layer (EL). With the successfully accomplished transition from PoW to PoS, there are some significant changes in the new reward mechanism of the multipurpose platform. Since validators now propose both blocks, they get the CL and the EL rewards. However, from the EL rewards, validators only earn the part coming from the transaction fees, which means that there is no base reward anymore. The fact that block proposers are chosen beforehand and the mining process has been removed increases the time to include transactions in a block. Which ultimately increases the chances of a proposer gaining more rewards by filtering and ordering the transactions.

This chapter of the Thesis presents a previously unseen approach to measuring the incentives transition that shifting from PoW to PoS had on a live-working blockchain. The proposed approach uses both empirical and theoretical methods to compare and score how well validators accomplish their duties. Moreover, we present some use cases where the methodology could be directly used to compare the performance of some staking entities based on their participation and rewards in the consensus layer.

5.2 Related Work

Blockchain-based networks adopted the peer-to-peer philosophy to consolidate platforms able to store a public ledger. Generally, these public ledgers use incentive systems to promote honest participation in the consensus. This way, for most users, honest participation in the network is more rewarding than misbehaving. Ethereum also uses the same system to promote the chain's finalization (reach a consensus on which chain to follow).

The unknown nature of block rewards in Ethereum's execution layer has attracted many experiments that tried to modelize it. Authors in [173] already introduced the apparition of mining pools for the Bitcoin [131] network. Pools where individual miners unified efforts to split rewards and costs, making them more stable over time. Other works like [181] presented an approach that, using linear and polynomial regressions, could predict the mining reward of the next Ethereum block to some degree. More focused on protocol optimizations, other works like [4] introduced an intensive study highlighting the under-optimized storage patterns that most smart contracts use. Furthermore, they propose a tool that helps optimize gas usage when developing smart contracts.

Showing a clear interest in predicting and maximizing the rewards extracted by each block, authors in [43] introduce the lack of fairness that decentralized exchanges and arbitrage bots introduced to the platform by using the usage of sandwich attacks [158] to front run users transactions. The authors present a novel idea of block-building audition or bid, introducing a set of techniques that optimize the reward extracted from each built block (also known as Miner-Extractable Value (MEV)), i.e., transaction ordering.

We have already presented in chapter 3, how the launch of the consensus layer and the shift of Ethereum to PoS, resulted in a way more sustainable network with over 900.000 validators distributed between 13.000 nodes in 90 different countries, and in chapter 4 the existing six different software that can adjust to all kinds of users' needs while increasing the overall resilience of the network to software bugs. However, despite the community's efforts to endorse a fully decentralized blockchain

[3] https://ethereum.org/en/history/#paris

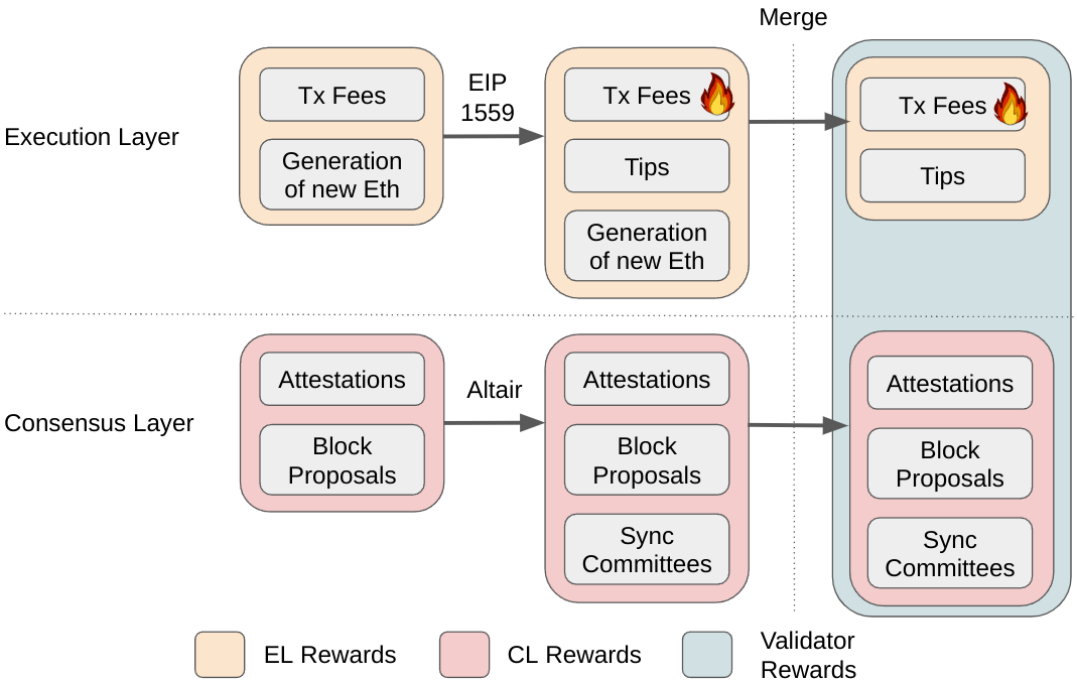

FIGURE 5.1: Evolution of rewards for different layers in Ethereum.

platform, we've identified a certain degree of centralization regarding geographical location and used software.

This chapter aims to complete the literature surrounding Ethereum's rewards from the consensus layer perspective. We present a novel method to evaluate the performance of validators by comparing their on-chain rewards with the maximum theoretical one they could obtain, identifying which are the most rewarding duties of the platform. We introduce the results of applying this method to Ethereum's transition from PoW to PoS during the merge. Furthermore, we present the decentralization degree that the current Ethereum network has by analyzing the distribution of validators among staking entities in the network.

5.3 Prime on Ethereum's Rewards

Economical remuneration is a highly used incentive to recompense honest behaviours in blockchain scenarios. However, building a sustainable economic rewards system with a solid game theory [116] requires considerable effort. This section explains the different reward types that Ethereum miners (previously) and Ethereum validators can earn from their honest interaction with the network, describing in detail the evolution of rewards displayed in figure 5.1.

5.3.1 Execution Layer's Block Proposal Rewards

To preserve a fair and controlled usage of the EVM, interacting with the Ethereum EL requires paying for the needed resources. Following its close analogy to vehicles needing gas to move, users pay Gas for their interaction with the EVM to the network processing it. Thus, each interaction with the EVM requires paying Gas for block space for the user's transactions. With the introduction of EIP 1559 [170] in the

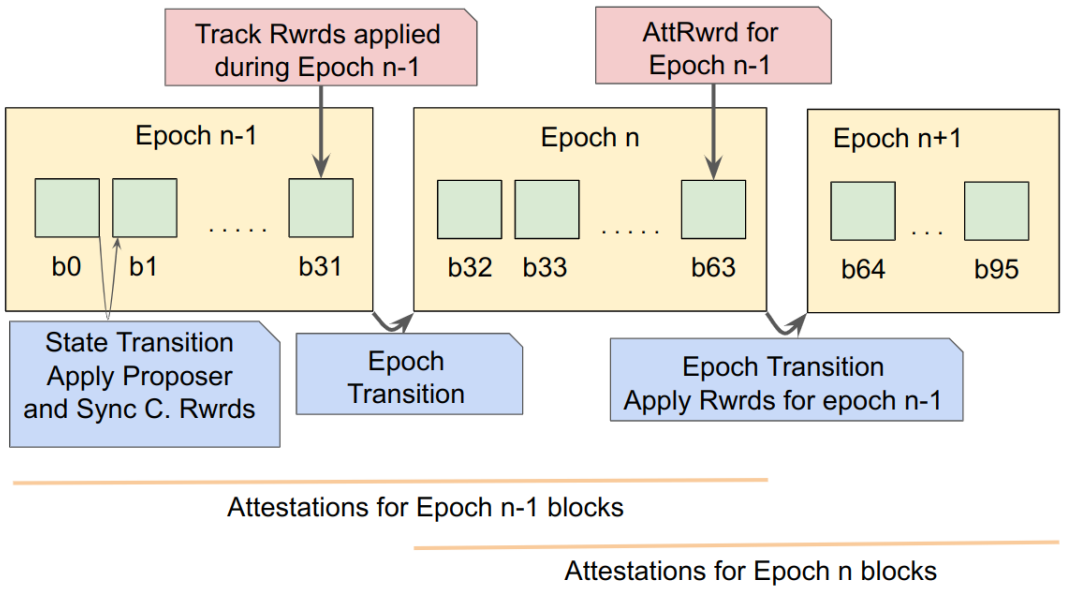

FIGURE 5.2: Rewards extracting schema followed by the state analyzer tool for Epoch $n-1$ (in red: actions of our indexer tool; in blue, actions at the protocol).

London Hard Fork[4] (see figure 5.1), the fees that a user pays to include a transaction in a block are divided into three different values:

1. Gas Limit: Upper limit of Gas that the entire operation may consume, protection against malicious or broken smart contracts.

2. Base Fee per Gas: Minimum price per gas unit to allocate a transaction in a block. Base Fee dynamically adjusts based on network congestion; the more demand for block space, the higher the Base Fee.

3. Priority Fee per Gas: colloquially known as the tip users pay to prioritize their transactions.

Leaving the total transaction as the product between the *GasLimit* and *BaseFee+Tip*.

In this new Gas system, block proposers in the execution layer no longer receive 100% of the Gas paid by the user. Currently, the aggregation of base fees included in each block is burned. As a result, since EL block proposals do not create new ETH in a post-merge scenario, EL block proposers only get the aggregation of the *Gas Tips* included in the block.

5.3.2 Characterization of Attestation Rewards

As part of each validator's duties in the network, once every epoch, all active validators are asked to participate in the consensus system by voting on whether a specific block is correct or not (understanding missing blocks as not correct ones). With this method, the network agrees and validates the data in the chain (finalization). As a result of performing this task, validators earn attestation rewards.

The whole set of validators is divided into beacon committees in each epoch of the beacon chain. However, each committee is assigned one slot per epoch. Thus, each validator is randomly assigned to attest to a single committee at a single slot,

[4]https://ethereum.org/en/history/#london

having a total of 32 slots to do it. After each validator has shared its vote with its respective beacon committee, all votes in that committee are then aggregated into a single attestation. If some validators send their vote later, another attestation is created with these votes, so no votes are left behind if they were sent inside a window of 32 slots (one epoch) after the slot they are attesting to. Moreover, the attestation or vote that each validator produces contains three main flags that determine the "quality" of the attestation:

- Source: the hash of the justified checkpoint[5] at the moment the attested block was proposed.
- Target: the hash of the first block at the epoch the attested block was proposed.
- Beacon block root: hash of the attested block.

After the *Altair's*[6] hardfork, each flag has a respective weight, determining the final reward per flag. Ultimately, the summary of the product between these three flags and their weights and a *BaseReward*[7], represent the attestation reward a validator will receive for its attestation to a given block.

From Ethereum's Consensus specs [63], the rewards for each attestation are directly calculated using the following formulas. Please note that for the sake of clarity in the chapter, we will summarize some of the aspects of the procedures to the basics (in the code implementation, for a better resolution between calculus, the formulas are calculated per units of the validator's effective balance).

$$BaseReward = \frac{EffectiveBalance * BaseRewardFactor}{\sqrt{TotalActiveBalance}} \tag{5.1}$$

$$FlagReward = \frac{FlagWeight}{WeightDenominator} * BaseReward * \frac{AttestingBalance}{TotActiveBalance} \tag{5.2}$$

$$AttestationReward = \sum FlagReward \tag{5.3}$$

Where: *EffectiveBalance* is the effective balance of the attesting validator, *BaseRewardFactor* is a constant of value 64, *FlagWeight* and *WeightDenominator* are constants that vary between flags, *AttBalance* is the sum of effective balances of all attesting validators, and *TotalActiveBalance* is the sum of the effective balance of all active validators.

5.3.3 Characterization of Sync Committees Rewards

Since Altair's hard fork, the Ethereum network has introduced the idea of sync committees, a group of 512 randomly selected validators who sign new block headers every slot. Light clients can use these headers to trust which blocks have been validated without fully downloading and processing the beacon chain. For a total of 256 epochs (8192 slots), each validator participating in a sync committee receives an extra reward: the sync committee reward. However, unlike attestation rewards, this reward is added at the state transition[8] of every slot, not at the epoch processing. After 256 epochs, the list of validators in the sync committee is refreshed.

[5]Checkpoints in the CL represent the Beacon State root of the epoch's first slot, including the result of the state transition from the previous epoch.

[6]https://ethereum.org/en/history/#altair

[7]Specific value that depends on the validator's effective balance and total active balance at a given epoch.

[8]The state transition corresponds to the act of including all the duties in the beacon chain in the parent Beacon State to produce the new Beacon State for the next slot. Includes balance modifications, slashing, etc.

For every slot that a validator correctly participates in the sync committee (by signing the new block headers and sharing this information with the other participants of the sync committee), its balance will be immediately updated at that slot. However, if the validator does not participate correctly, it will suffer a penalty for missing its duty. The sync committee rewards per block can be summarized as follows:

$$TotalSyncCommitteeReward = TotalBaseReward * \frac{SyncRewardWeight}{WeightDenominator} \tag{5.4}$$

$$IndvSyncCommitteeReward = \frac{TotSyncCommitteeReward}{SlotsPerEpoch * SyncComSize} \tag{5.5}$$

Where: *TotalSyncCommitteeReward* is the sync committee reward for the whole committee per epoch, *TotalBaseReward* is the sum of the *BaseReward* of all the active validators, *SyncRewardWeight* and *WeightDenominator* are constants related to the sync committees, *IndvSyncCommitteeReward* is the individual reward per validator per slot, *SlotsPerEpoch* is a constant of 32 slots per epoch, and *SyncComSize* a constant of 512 validators per sync committee.

5.3.4 Characterization of Block Proposals

For every epoch, a total of 32 validators are randomly chosen to be block proposers. Each block proposer will have the duty of proposing a block in one of the slots of the epoch. In the opposite direction to the constant and stable attestation rewards, block proposal rewards are way more sporadic due to the randomness of the block proposer election. With a current total of over 900.000 active validators in the network, the chances of being elected as a block proposer are around one every four months. However, the rewards linked to proposing a beacon block are high. Those rewards are coming from:

- Including the attestation aggregations from validator votes that are not yet included in the beacon chain.
- Including sync aggregates. The block proposer is in charge of including the sync aggregate from the 512 sync committee participants.

Similarly to sync committee rewards, proposer rewards are added to the validator's balance when the slot (and, thus, the block) is processed, being the formulas to calculate the proposer reward per each block the following:

$$AttestationRewards = \frac{\sum BaseReward * FlagWeight}{\frac{(WeightDenominator - ProposerWeight) * WeightDenominator}{ProposerWeight}} \tag{5.6}$$

$$SyncCommiteeRewards = \sum \frac{IndvSyncCommitteeReward * ProposerWeight}{(WeightDenominator - ProposerWeight)} \tag{5.7}$$

$$ProposerRewards = AttestationRewards + SyncCommiteeRewards \tag{5.8}$$

Where: *BaseReward* corresponds to every validator whose vote was included in the block, *WeightDenominator* and *ProposerWeight* are constants related to the proposer rewards, and *IndvSyncCommitteeReward* is the individual sync committee reward for each sync committee attestation included in the block.

5.3.5 Validators' Penalizations

As an incentive to finalize epochs, Ethereum PoS validators receive rewards for their contributions when they are consistent with 66% [25] of the total validators (reaching consensus). However, they can also be penalized when they don't comply with their duties or when their duties don't match with the majority of validators. Penalties are the only protection mechanism that the platform has to defend the finality of the chain against bad actors or long time disconnections, ensuring that eventually, the network will be able to finalize again because the balance of those actors ends up below the minimum one needed to participate in the consensus. In terms of quantifying the penalties, the PoS consensus mechanism penalizes validators when they miss-perform critical duties summarized as follows:

- When a validator misses attestation flags, only the *Source* and *Target* flags will penalize the validator's balance. The quantity of balance that will be deducted equals the reward it would get for each flag but in negative. This ensures that one day of correct attestations means more rewards than the deducted ones for an inactivity day.
- Missing block proposals are bad enough for the validators as they represent a high reward in a single slot. The rest of the network is in charge of attesting for an empty slot. Thus, no extra penalty is applied to the validators that missed a block proposal.
- The network doesn't have much mercy on validators when they miss a sync committee duty. Because they are essential for light clients to verify the validity of the latest epochs. The entire reward the validator could have had will be deducted from its balance when sync committee duty is missed.

The previously introduced penalties refer to the duties of a validator that were, according to the consensus, wrongly done or missed. However, there is a set of actions a validator could easily do that could create conflicts inside the network. For this reason, the consensus protocol has a set of rules that, whenever any validator breaks them, it is subject to slashing. The slashing can be triggered when a validator performs the same duty two times, i.e., voting with two different attestations to the same slot, as it can divide the network in different forks depending on which attention was received first. The slashing of a validator presents the exit of the validator from the list of active ones and three concurrent balance modifications:

- The penalization of the validator whose balance gets decreased:

$$SlashingPenalty = \frac{EffectiveBalance}{32} \tag{5.9}$$

- The reward for the validator that proposed the block that includes the slashing, whose balance increases:

$$ProposerSlashReward = \frac{EffectiveBalance * ProposerWeight}{512 * WeightDenominator} \tag{5.10}$$

- The reward of the *Whistleblower* that reported the incident, whose balance increases:

$$WhistleBlowerReward = \frac{EffectiveBalance}{512} - ProposerSlashReward \tag{5.11}$$

5.3.6 Maximum Extractable Reward (MER)

The presented formulas in the above sections describe Ethereum's current reward system when writing this chapter (post-Paris Hard Fork[9]). As a reminder, for a healthy performance of the beacon chain (keep finalizing states), only 66% of the validators actively need to participate in the consensus on new blocks. This means that even though the network achieves resilience to validator churn[10] the network is still able to finalize states in the chain. We understand as participating validators, the ones that are actively attesting in their committees; however, in the last instance, all the rewards that are attesting validators, sync committee members, and block proposers receive also depend on the quality of their duties (i.e., an attestation might have 1/3 flags correct and still be considered as participation to the consensus).

This said, and as part of our contribution to this chapter, we identified that we could theoretically qualify and compare the maximum reward each of the validators could achieve at every given epoch with the one they actually achieved. From now on, we will refer to this theoretical max reward as the Maximum Extractable Reward (MER).

5.3.7 GotEth: Ethereum Chain Indexer

Despite all of Ethereum's consensus data being publicly available, interacting with it is not trivial. As introduced in chapter 4, the only possible interaction with the chain data is through the usage of the beacon nodes' REST API. As the number of endpoints provided at the REST API is limited[11], a restricted number of data can be directly obtained from them. Thus, to obtain the consensus interaction of validators, we must aggregate the raw data obtainable through the basic REST API endpoints.

As the Ethereum research ecosystem is still growing as more people join it, at the moment of writing this chapter, there wasn't any chain indexer software except for *Chaind*[12], which suffers from a few big drawbacks such as:

- Despite allowing the indexation of a specific range of chain slots, the indexing process is unstable and reasonably slow. Moreover, we encountered many sudden crashes during the full chain indexation from Genesis.
- Despite allowing the selection of which specific metrics to index in the database, the inability to aggregate metrics before persisting them to the database forces the indexation of more raw metrics and tables than needed. This heavily increases the storage size of the database.
- The inability to perform calculations or aggregations over metrics at the moment of indexation forces to generate extra software in a post-synchronization stage.
- Even if all the metrics were indexed, ignoring the sudden crashes and large storage requirements, the aggregation of metrics after combining multiple tables suffers from significant time delays in processing the SQL queries.

Due to this lack of existing chain metrics aggregator tools, we developed an open-source tool called *GotEth* [125]. The tool aims to cover most of the drawbacks

[9] https://ethereum.org/en/history/#paris

[10] Validators that go offline for some time and return afterwards (due to client updates, internet shortage, among others).

[11] Beacon API's standard: https://ethereum.github.io/beacon-APIs/

[12] https://github.com/wealdtech/chaind

presented by *Chaind*, allowing the user to choose between the level of metrics it wants to store, keeping only the aggregated or processed metrics. *GotEth* can index the following information per epoch:

- Individual duties per validator and/or per set of validators (who had to do what during the epoch)
- The quality of these duties, i.e., if validators missed a block proposal, the number of flags successfully voted, etc.
- The balance difference of each validator between epochs, composing the reward they got through their duties. Furthermore, it splits the obtained reward based on the duty that generated it.
- A validator could get a maximum attestation and sync committee reward on an epoch. Composing the MER.
- Can also index the Execution's payload, including total gas used per block and transaction, the paid fees by users, and the raw transactions.

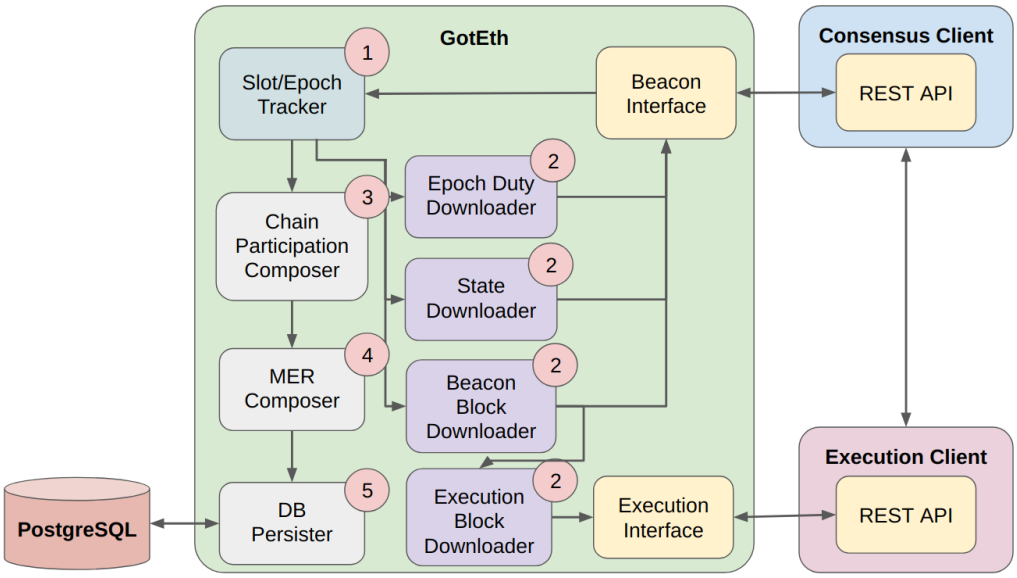

FIGURE 5.3: Internal diagram of the Ethereum chain indexer tool. In green, the internal structure of the tool showcasing the five main steps: 1) identify when each epoch and slot starts, 2) download of the epoch duties, the beacon block on each slot (if it exists), the state at the end of the epoch, and the execution receipts, 3) with the raw chain data, compose the participation of each validator and the reward that each one got, 4) based on the chain state, compose the MER of each validator, and 5) persist of the metrics into the database.

Figure 5.3 shows the internal diagram of *GotEth*, where we can also appreciate the necessary deployment to run it. As previously mentioned, *GotEth* obtains chain data from the interaction with a beacon node and an execution node. Due to the heavy usage of the API and the delay that some endpoints, like fetching the beacon state, could add to the process, we highly recommend running the beacon node in "archival mode" 4.3.2.

The first module of the tool is a slot and epoch tracker. The tool can either index a range of slots or follow the canonical head of the chain. However, in both cases, a

specific module is needed to define the slot we are trying to index. With each slot, the tool triggers a set of queries to the consensus and execution nodes, fetching also the entire Beacon State[13] when the slot defines the end of an epoch. Once the tool knows which duties each validator had to perform, it begins to process the accomplishment of the same ones and, thus, the reward each one got in return.

Ethereum's consensus has a unique way of processing and applying rewards, providing some time ranges for validators to submit or include attestations. The consensus contemplates that attestations can be included in 31 slots after the block they are referring to, and they are only applied to the balance of validators once the state transitions have been applied when a new epoch starts. As a visual reference, the schema displayed in figure 5.2 shows how *GotEth* tracks the rewards obtained by validators on each epoch. The tool chronologically analyzes the states, tracking the proposer and sync committee rewards applied during epoch $n-1$ at the last epoch slot (slot 31 in the figure). Once it has already distinguished the rewards, it processes the next state, the state of the last slot in epoch n (slot 63 in the figure). From the second state, it analyzes until the end of the given time range. In the last slot of epoch n, the tool can already measure which are the duties that were accomplished concerning epoch $n-1$ and which ones were not, which lets us rate how well each validator performed and which would be its maximum reward for that particular attestation (composing the MER for each validator on each epoch). Although it might be a bit misleading, these attestation rewards we can already compute won't appear in the validator's balance until the epoch transition is applied.

To finalize the processing of an epoch, the tool persists in a PostgreSQL database with the corresponding metrics, indexing the metrics in distinct SQL tables. This heavily eases the data's post-processing to analyse a validator's performance and profitability on a given epoch or period.

5.3.8 Validator to Staking Pools Mapping

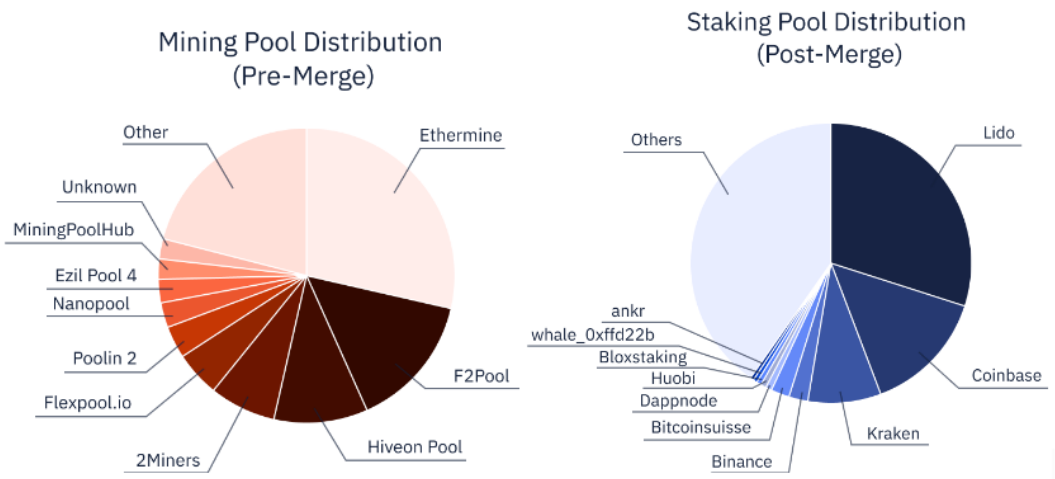

FIGURE 5.4: Block proposal's distribution per mining and staking entities for the two months of study.

We have already introduced the steps and requirements to become a validator in Ethereum's PoS. However, collecting 32 ETH (for optimal performance) and running

[13]The Beacon State is the primary data structure that represents the beacon chain's status at every slot, including the entire list of validators, their balances, and duties.

the required software 24-7 at home is impossible for every user. Either the lack of funds or technical knowledge to set up and maintain the validator online creates a large opportunity for delegated staking and the generation of staking pools. As it originally was for PoW with mining pools, the concept of staking pools consists of aggregating multiple users' funds until you can activate a validator. This type of service is generally driven by entities or organizations that are in charge of activating the validator, setting up the clients, and maintaining it.

In the Ethereum ecosystem, there are many entities. There are centralized organizations such as Coinbase[14], Kraken[15], or Binance[16], and decentralized entities like Lido Finance[17] or Rocket Pool[18], which allow users to delegate their staking. To differentiate the validator performance of the wide set of staking pools, we mapped the whole set of validators to their respective staking pool (only for those linked to a staking pool) based on the transaction that activated the validator using the PoS smart contract[19]. Since activating the validator in the Beaconchain requires interacting with smart contracts from a "deposit address", we inspected the whole list of PoS deposits, mapping (if possible) the "deposit addresses" to any of the Ethereum addresses publicly identified from the entities. Figure 5.5 shows the result of classifying the entire list of validators based on the deposit addresses or the publicly available data.

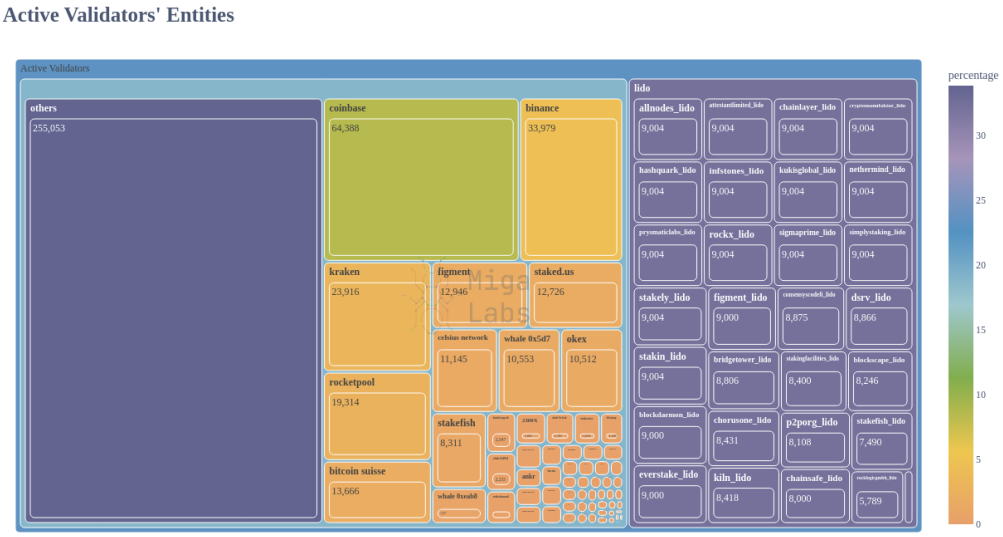

FIGURE 5.5: Distribution of active validators across the known staking entities at 16th of February 2024

[14]https://www.coinbase.com/
[15]https://www.kraken.com/
[16]https://www.binance.com/
[17]https://lido.fi/
[18]https://rocketpool.net/
[19]More about the methodology can be found in this GitHub repository: https://github.com/migalabs/eth-pokhar

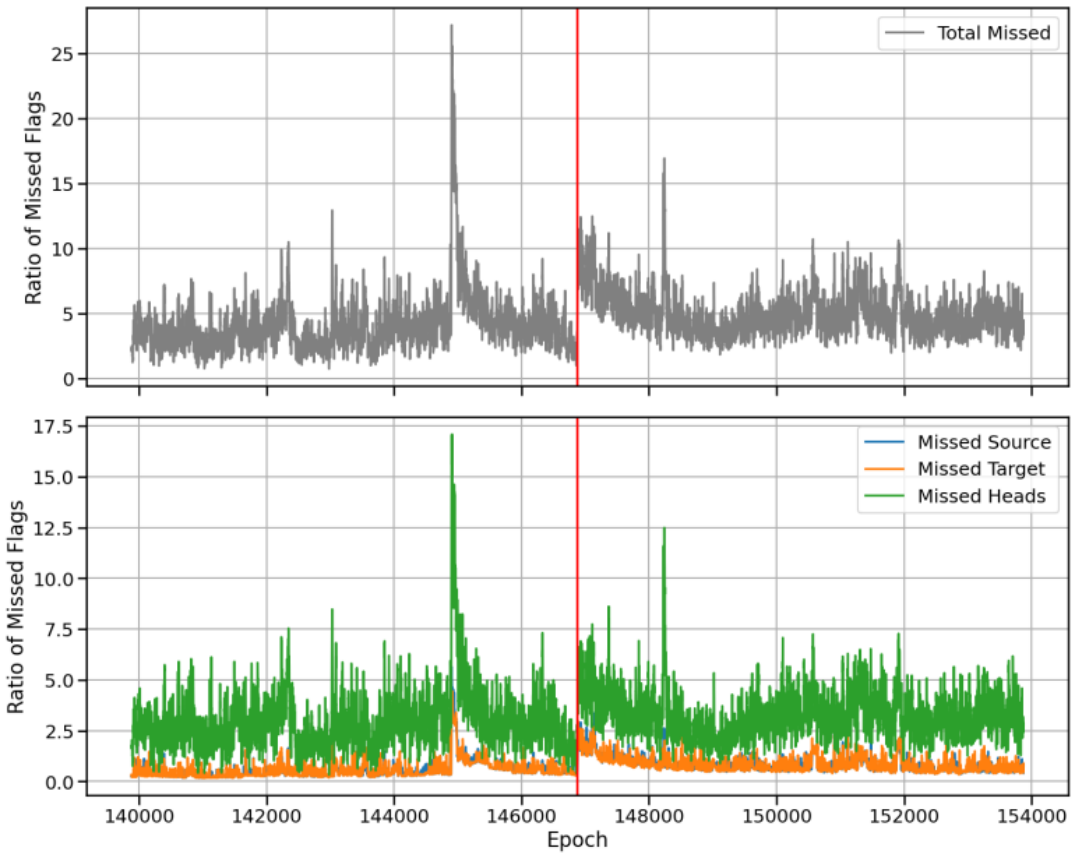

FIGURE 5.6: Time distribution of missed flags percentage from the total number of validators. Aggregating the total missed flags on the top chart and distinct missed flags at the bottom.

5.4 Performance of Validators during "The Merge"

After identifying the main components of Ethereum's incentive systems and the concept of MER, which can help us understand how well a validator performs, this section introduces the insights of analyzing the performance of all active validators between epochs 139.875 and 153.875[20]. These dates represent precisely one month before the merge and one month after, which helps us understand the impact of the transition to PoS on validators' performance.

5.4.1 Descentralization of the Consensus

Combining efforts to support the healthy performance of the network is a key point of the Web3 space. We have seen how individuals were uniting efforts to remain competitive while supporting decentralization during PoW. PoS is not different in that sense. We analyzed the level of decentralization pre- and post-merge to determine if the platform improved or got worse in the aspect of decentralization. For that, we looked into the block proposal distribution for the ten largest entities: mining pools for the pre-merge (determined by fee recipient and using Etherscan [65]) and staking pools and entities' validators for the post-merge. Figure 5.4 summarizes

[20]This chapter presents the main findings of the study, focusing on a larger picture of the overall rewards, and presenting the examples of the largest staking entities. However, all the data is publicly available at https://pandametrics.xyz/

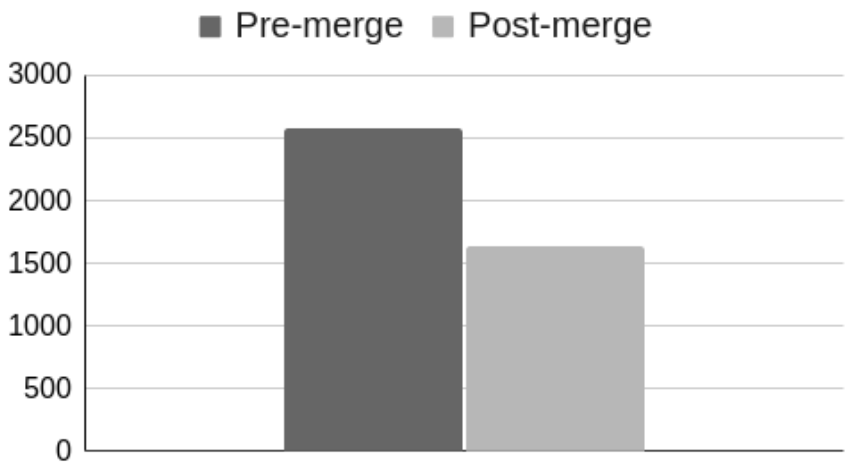

FIGURE 5.7: Number of missed blocks pre- and post-merge.

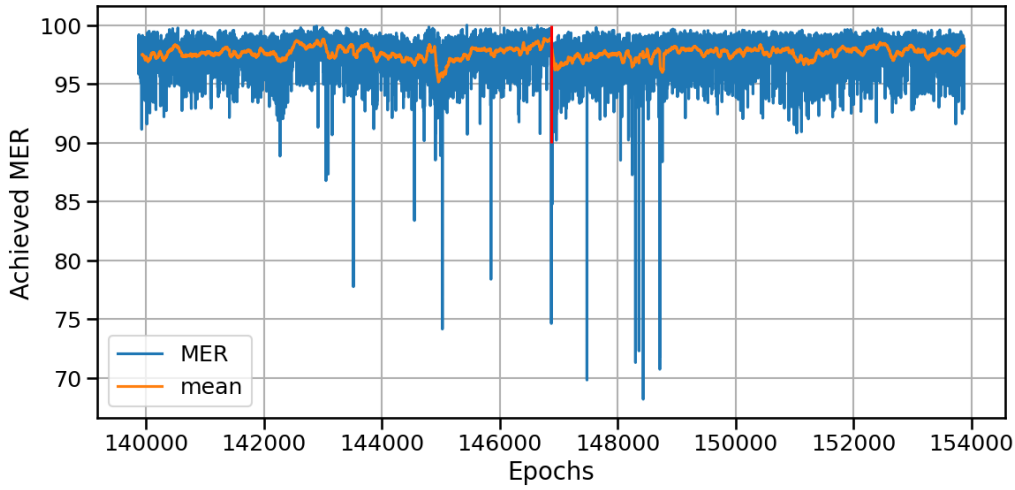

FIGURE 5.8: Maximum Extractable Reward for validator network over the 14k epochs.

that distribution, where overall, we observed a rather similar decentralization level. For instance, pre-merge, we see that Ethermine, F2Pool, and Hiveon Pool proposed 28.5%, 14.7%, and 10.2% of the blocks, respectively. Post-merge, Lido, Coinbase, and Kraken proposed 29.9%, 14.2%, and 8.4% of the blocks. Nonetheless, it is essential to keep in mind that Lido consists of 28 operators[21] that are somewhat independent, which adds another level of decentralization. In addition, we observe notably more small-scale/solo staking ("Other" category) participation (36%) in comparison to small-scale/solo mining (20.9%), likely due to the reduced infrastructure and energy requirements that PoS represents over PoW.

5.4.2 Validators Overall Performance

As previously introduced, achieving consensus and the finalization of the chain now depends on the contributions of validators' duties. To rate the performance of validators and ultimately monitor the chain's performance, we looked at the behaviour

[21]The Lido DAO governs Lido and provides guidelines to node operators, such as increasing client diversity and using risk-management techniques; and a policy, such as how operators are expected to use MEV-Boost.

of the overall CL network (CL validators now propose EL blocks). Before making specific comparisons between staking entities, we analyzed the total missed duties of the active validators.

Missed Attestation Flags

Regarding the validator's attestations, figure 5.6 displays the distribution of missed flags for the 14.000 measured epochs (note the red vertical line that marks the merge epoch). The upper chart in the figure represents the aggregation of all the missed flags normalized by total active validators per epoch. In contrast, the one at the bottom represents the same normalized distribution but is split between each of the flags present in the attestations.

We observe a slight increase in the total of missed attestation flags, going from a 3.9% pre-merge to a 4.9% post-merge. Considering the different flags included in the attestation, the numbers go from 0.5%, 0.5%, and 2.9% pre-merge to 0.8%, 0.8%, and 3.3% post-merge for *Source*, *Target*, and *Head*, respectively. In the upper graph of figure 5.6, we can also observe two distinct patterns:

1. A significant spike of 27.17% of missed flags around epoch 145.000, which we attribute to a large number of restarted clients that were updated to support the merge.

2. A longer time range of total missed flags between epochs 146.875 and 148.000 that we attribute to those validators underperforming or missing duties for not having upgraded their software.

Moreover, we can appreciate a large difference in the number of missed flags inside the attestations. In the chart at the bottom of the figure, we can appreciate the validators tend to miss the *Head* flag more often than the *Source* and the *Target* ones. The chart also shows that the numbers between the *Source* and the *Target* often match. Meaning that if a peer is missing the *Source* of the attestation (the hash of the justified checkpoint), it is quite likely to miss the *Target* and the *Head* of the attestation. In most cases, missing the three flags in the same attestation can also be interpreted as a "not attested" sign, which a non-operative client or a non-fully synchronized one might cause.

MER

The economic incentive promoted by most of the blockchain platforms directly bounds performance and financial rewards. Ethereum is not different in that aspect; validators are rewarded with ETH tokens for their successfully accomplished duties. Figure 5.8 displays the percentage of the MER obtained by the network at each epoch (from attestations and sync committees rewards). We can observe in the figure that the mean for the MER achieved oscillates between 95% and 99%, with some sporadic drops that can reach 68% of the MER at some epochs. However, it also means that some validators miss the chance to earn more rewards in each epoch. As previously introduced in section 5.4.2, we attribute these drops to an offline or syncing CL client.

Missed Block Proposals

When comparing the number of missed blocks for $7K$ epochs pre-merge to $7K$ epochs post-merge, figure 5.7 shows an absolute value reduction of 36.40% in missed blocks.

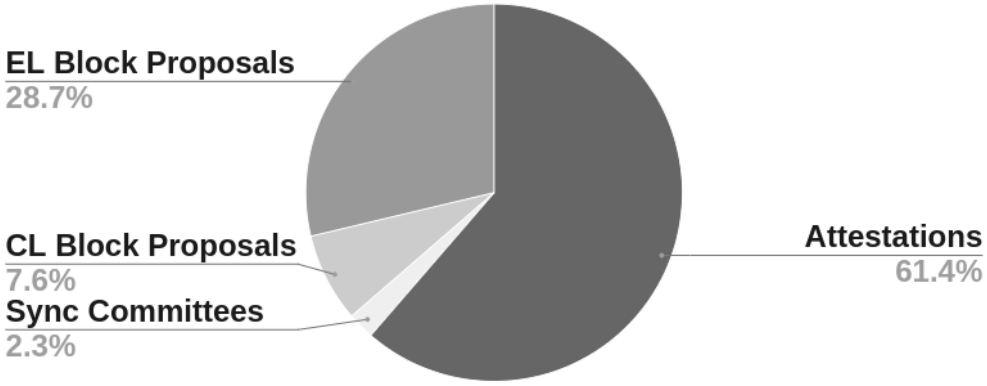

FIGURE 5.9: Empirical distribution of rewards from different sources.

We relate this reduction to the fact that missing a block post-merge has a double economic penalty for not getting either CL proposal rewards or EL rewards. Thus, one could expect validators to make every effort to avoid missing block proposals. However, we have to remark that even in the pre-merge scenario, the number of block proposals missed represents 1.13% of the total number of blocks. So, the actual 36.40% of reduction refers to a reduction from a 1.13% to a 0.72% ratio of missed blocks.

5.4.3 Empirical Distribution of the Rewards in a post-Merge Scenario

From the advantage of indexing all the possible rewards, we have been able to track and compute the different origins for all the validators in the network, and figure 5.9 displays all of them. The chart shows that 71.3% of the rewards for a validator are generated through the new consensus layer. Digging a bit more into the CL rewards, the figure showcases that 61.4% of those total rewards are coming attestations, which coincidentally are the most stable rewards a validator can have. Leaving the block proposals in second place with a 7.6% of the total generated rewards, 28.7% for the EL block proposals coming from transaction fees or MEV[22], and 2.3% for the sync committee rewards.

5.4.4 Concatenation of Multiple Block Proposals

As long as the beacon chain keeps finalizing, more validators keep joining the network. With a continuous linear increase of 3.8% validators over the two months, the chain went from 428.000 validators at the beginning of the pre-merge period to 444.000 validators at the end of the post-merge period. With this increase in the total number of validators, the chances of being a block proposer decrease. However, the randomness of the RANDAO algorithm still provides some luck to a few validators that can propose multiple blocks in a time range of 2 months. Table 5.1 represents the frequency at which individual validators propose blocks, where we can observe that 90% of validators proposed either one or no blocks. However, a few lucky proposers still proposed 6 or 7 blocks in the same short period. These findings closely

[22]Miner-Extractable Value (MEV) is a new mechanism to compose EL blocks. Block builders compete with each other to build the most rewarding block for the block proposers (maximizing the usage of gas per block, organizing transactions prioritizing tips, or through private transaction channels.)

TABLE 5.1: Table with the number of validators proposing blocks consecutively. (Pre-merge and Post-merge comparison)

	0	1	2	3	4	5	6	7
Pre	255,983	131,192	34,145	6,088	805	90	5	1
Post	270,251	134,458	33,682	5,747	739	62	5	-

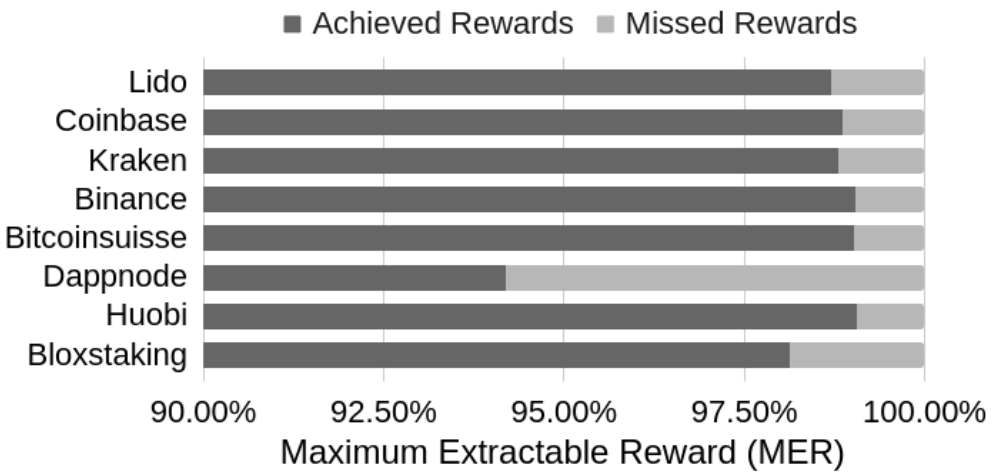

FIGURE 5.10: Maximum Extractable Reward for the eight biggest staking entities.

align with the fairness rewards study [85] performed over different consensus incentive systems, where authors showcase the steady fairness that compounded PoS incentive models, like Ethereum, provide among their participants.

We have already introduced the origination of staking entities in the space, accumulating thousands of validators. As we already saw, individual validators have chances of proposing several blocks in a short period. Extrapolating those chances to staking entities' validators, there isn't only a higher number of block proposals in short periods but many consecutive block proposals for the entities. Table 5.2 shows the number of consecutive blocks proposed for the three biggest staking entities (starting from the 4th consecutive proposal for space reasons). The table shows that Lido, with 27% of the validators, proposed four consecutive blocks 3686 times in only two months. Achieving even ten consecutive block proposals (a third of the epoch) on five occasions. Despite each validator eventually gains the "same" reward as the rest, the uneven token distribution in the network still allows a certain selected group of individuals or entities to accumulate more validators. As works like [101] present, Ethereum's ETH coin shares a Gini coefficient of 0.499 among the top 100 token holder addresses, outstanding all compared ERC20 token-based blockchains (0.685-0.907) and remaining close to the BTC Gini coefficient (0.466).

Staking pools or entities have a legitimate place in the ecosystem; they allow staking for users who would not participate otherwise. However, since the merge happened, having this larger set of chances to propose consecutive blocks could mean a hazard for the overall network. If a large staking entity had some dishonest interests, it could be benefited by allocating users' transactions not only in the order that they want (i.e., performing a sandwich or similar attacks) but to allocate the transactions in the block they want, giving them a higher time window analyze the

TABLE 5.2: Table with the number of consecutive blocks proposed by the larger staking entities.

Operator	Merge-Stage	4	5	6	7	8	9	10
Lido	Pre	1849	529	169	51	23	2	5
	Post	1837	524	168	48	23	2	4
Coinbase	Pre	145	18	4	-	-	-	-
	Post	144	17	4	-	-	-	-
Kraken	Pre	20	3	1	-	-	-	-
	Post	20	3	1	-	-	-	-

public transactions' pool. In the ultimate instance, the only thing that prevents large entities from adopting a dishonest position in the network is their public exposure to the same one, as validators are publicly identified in most cases.

5.4.5 MER per Staking Entities

In previous sections, we have already analyzed the entire network's performance. However, in a scenario where most users want to delegate their stake to others, figure 5.10 displays the achieved MER percentage for the eight biggest pools in the network. Considering that MER (max attestations and sync committee rewards) represents 61.4% of the total rewards of the two months we analyzed, we observe that some of the largest staking pools do not fully achieve a portion of the MER. From a minimum of 94.2% for Dappnode to a maximum of 99.07% for Houbi, an average loss of 1.77% of the MER shows a high level of commitment from the respective entities. However, the absolute value of ETH tokens that 1.77% represents for those entities is still relevant.

5.5 Conclusion and Future Work

This chapter presents the shift in the rewards system that "The Merge" implied for the Ethereum platform. We have unveiled that in the new incentive program that the platform proposes, the consensus layer's rewards represent 71.3% of the total rewards, showcasing the importance of having well-behaved and active validators that achieve 61.4% of their rewards from constant and stable attestations rewards.

The presented study analyzes the merge of both chains by tracking the missed duties of validators during the process. Overall, the network keeps a healthy behaviour that maintains a low ratio of missed blocks and missed attestation flags of 0.72% and 4.9%, respectively. We have identified several drops when comparing the obtained CL rewards with the MER during the merge, highlighting some instances where the network miss-performed, achieving 68% of total possible rewards at some point. Moreover, the chapter exposes that, currently, the platform faces a high ratio of consecutive blocks being proposed by large staking entities. We introduced that there were even several cases where a third of an epoch was proposed consecutively by a single entity. We have identified the hazard this represents due to these entities' control over which transactions get included and when.

Ultimately, the chapter introduces our rewards measuring methodology, which can be used to monitor the performance and healthiness of any set of validators in the network. We have observed that, even though big staking pools show a high

commitment to the finalization of the network, they still miss 1,77% of the rewards generally due to a non-optimal performance on their clients.

As future work to the presented methodology and study, we aim to extend the methodology to include important protocol upgrades and protocol misbehaves. This update includes the addition of an MEV tracking to measure the 100% of the Execution layer's rewards that a validator perceives and the adjustment of the MER to the possibility of facing a missing block (which forces the committees of that slot to fail the "head" flag). Furthermore, we would like to explore the different causes that make validators miss the flags of their attestations and extrapolate the presented rewards study to compare the different existing software.

Chapter 6

Impact of the P2P Network on the Ethereum's Decentralization

As introduced in chapter 3, any blockchain application is subjected to the state and healthiness of its peer-to-peer network. The distributed nature of the protocol relies on the network's capabilities to stay peered and its ability to quickly and efficiently propagate each of the validators' duties. Section 2.3.2 introduced the small time margins that the transition from PoW to PoS has meant for the Ethereum consensus, where a one-second delay can define the acceptance or rejection of a block proposal, or the margin to define whether an attestation flag is correct or not.

Under such time-limited windows that Ethereum's consensus gives, it is still unclear which role the network takes. The performance of a validator has to be linked with the healthiness of its connectivity. However, to our knowledge, this topic hasn't been researched, quantified, or modelized yet. There aren't any clear relations between which parameters impact the performance and what actions can be taken to overcome them. Moreover, it is still unclear how the protocols get affected by those parameters, i.e., if the node's connectivity impacts the profitability of a validator, is there any hidden incentive from the network to cluster on low-latency connected geographical regions?

The existing lack of literature linking network connectivity to blockchain's application performance, beyond the model presented in this thesis' chapter 5, makes it hard to judge whether the proposed protocol upgrades had any side effects that could alter the finalization of the chain. As previously mentioned, the tight time division of a slot is currently a limiting factor of what can be done to improve the protocol, i.e., the extra delay of increasing the Execution Layer's rewards through the usage of MEV is already messing up in some occasions the block proposals.

In the absence of answers to these questions in the existing literature, this chapter, as the core of the thesis and the combination of all the previous chapters, presents the direct impact or limitations that Ethereum's peer-to-peer network has with the slot time division on the profitability of validators and the on the resilience improvements proposed by the Distribute Validator Technology (DVT).

6.1 Unveiling Ethereum's Hidden Centralization Incentives: Does Connectivity Impact Performance?

6.1.1 Introduction

The successful transition from PoW to PoS of Ethereum [93] implies a radical change in the consensus mechanism. Validators must actively participate in the consensus

to keep finalizing previous epochs, assigning them periodical duties they must accomplish over their lifetime. However, to perceive the maximum remuneration that PoS can grant to honest participating validators, the quality of their implication has a significant weight on their reward. Each validator has the following duties to fulfil: i) attest on a slot every epoch (defined randomly on the state transition between epochs), ii) sign sync-committee duties if the validator belongs to a sync-committee, and iii) generate and propose blocks when they are chosen to do so. The validator is now in charge of proposing blocks when they become block-proposers.

All these duties and their implication in the consensus are defined in the Ethereum specification [63]. However, although there is one single specification, Ethereum relies on a wide variety of implementations to introduce the resilience of the protocol. As chapter 4 introduced, six main clients are consolidated to participate in the network: Lighthouse, Lodestar, Nimbus, Prysm, Teku, and Grandine (although the last one remains closed-sourced). Each uses a different programming language to implement the networking and consensus specs. Even though this multiple-spec implementation significantly impacts the feature development time (extra complexity making the implementations inter-operable between clients), it makes the protocol fault-tolerant. Ensuring that with proper client diversity in the network, a bug on a single client won't break the aggregation of new blocks to the chain.

It is inevitable, however, that although all the implementations are spec-compliant, the wide variety of conditions in the protocol makes some algorithms more optimal than others under specific circumstances, i.e., certain network instabilities or the resulting latency delay derived from the geo-location of the node. The chosen algorithm to aggregate attestations or any code optimization that prepares some resources before a given event occurs might imply better participation in the consensus. As chapter 5 introduced, having a faster or wiser algorithm might get mirrored in a more accurate duty performance, which could directly impact the rewards of the validators.

Although the ecosystem is aware of software differences and the networking conditions, the impact of these parameters on the performance and profitability of a validator has only been tested on a private enterprise level. Due to the lack of transparency on the setups, the methodology, the results, and external parameters such as technical preferences towards a specific programming language, there is a research and community interest in addressing and quantifying the impact of the setup on the validators' performance. Thus, this chapter of the Thesis focuses on identifying and quantifying the performance differences between geographical regions from a CL client perspective, with the final intention of spotting any existing relevant performance difference that could compromise the client diversity in the network.

From the premise that a better accomplishment of duties generally means more reward for validators, the chapter analyzes the direct implications of the networking conditions on the stability and performance of the five main CL clients. We present a study that measures the duties completion of multiple clients across multiple locations in live networks, which has not been previously done to the best of our knowledge. The contribution of this chapter was to identify any missed performance between different geographical locations that could compromise the network's decentralization while proving that client diversity in the network is mature enough to be achieved in production without significant sacrifices.

6.1.2 Related Work

Distributed ledgers are an interesting phenomenon in the internet space. In such a critical environment where users trust the network to track their economic balances, participants must agree on a set of rules to reach a consensus over the interaction with the ledger for personal and shared interests.

From the nature of distributed networks, peers can join, leave, and disconnect the network as they please [87, 99], i.e., users turning off their nodes to upgrade their version. This shows that peer-to-peer networks dynamically adjust and react to events over time. Changes in jurisdictions or local regulations can make the network pivot from one region to another. An example of this was the radical legislation of the Chinese government over Bitcoin mining [91, 165], which forced a big part of the Bitcoin network to migrate to operate from other parts of the world.

Chapter 5 introduced the incentives validators have for actively participating in consensus. As the network is deployed to keep these validators running, PoS blockchain networks are more stable in general. Nonetheless, they still represent Byzantine fault tolerance systems [30], where the system can overcome at some degree the sudden decrease of the honest participants, ensuring a unanimous consensus over the state of the chain.

On the operational side, PoS provides an easier migration of the operating location. It is more effortless with cloud service providers, which makes spawning another machine in a different region as easy as a click [95]. However, on an existing network highly congregated in a relatively short list of countries whose population can afford activating a validator, the impact of this network clustering on the performance of validators and, thus, on the better viability of nodes in an already populated network still remains unknown.

The previous chapter 3 showed Ethereum CL's network topology, introducing some concerns about this topology, identifying a concerning client centralization on two software (Lighthouse and Prysm) and geographical centralization in two countries (US and Germany). Although we perceive a healthy network behaviour, some previous studies like [97] have demonstrated that the node's location directly affects the networking performance and, inevitably, the performance of PoW-based application's nodes.

Message distribution is essential in PoS systems such as Ethereum. Although each slot gives a time window of 12 seconds to propose a block, commit the attestations, and aggregate them, receiving half a second sooner or later the block can directly impact the attestation of validators, as their attestation could shift from the block valid, to there wasn't a block at all. This means nodes in regions far from the network's core could be disadvantaged. Placing a client in a poorer connected region can incentivize other validators to keep concentrating on the exact geographical locations. Suppose a latency increase can risk a validator's performance (and the economic stimulus attached to it). In that case, the short window of action that the protocol suggests would incentivize the centralization of clients in regions with the counterpart, increasing the exposure to local authorities' censorship.

This chapter will present the Ethereum CL performance comparison results based on network latency in different locations. In the study, we empirically analyze the stability of the clients under the fundamental network behaviours of Ethereum's mainnet and the Goerli testnet. We analyze the performance of the different implementations by comparing the quality of the accomplished validator duties comparing the profitability of the blocks generated by the other implementations. Furthermore, we reproduced the experiments in different geographical locations to explore

the impact of the message propagation latencies on the ability to perform consensus duties.

6.1.3 Methodology

Two big pillars of public blockchains are the transparency and immutability of the public ledgers they compose. This means that as data gets added to the chain, the information is publicly available when there is a trusted connection with a synced node. Since the apparition of the beacon chain, the chain doesn't only keep balances of addresses, smart contracts, or transactions that interact with both. The beacon chain also stores the validator's aggregated attestations, their flags, the balance increase between epochs and slots, etc.

Chapter 5 presented how the chain state can be reconstructed by accessing the stored data in an Ethereum full node as long as it was proposed and accepted by the network. However, this methodology doesn't contemplate that some occurring events might not arrive on time to the 66% of the network, therefore, not getting accepted. This is easier to represent by taking Ethereum's public "Mem-pool" as an example, the local list of transactions that each execution node keeps in memory for when it has to compose an execution payload for a new block. Although a given transaction has been broadcasted to the network, thus being present on many local "Mem-pools", it isn't accepted until it has been included on a block and then accepted by the network. There is no guarantee of getting the transaction included in the coming blocks [9], which means we could be missing the track of non-included transactions if we don't monitor the "Mem-pool" with a short frequency. Due to the lack of inclusion guarantees of the system, this one is prone to malicious users that could potentially generate censorship attacks over transactions [111] or even the consensus [210].

For the case study of this Chapter, where we want to empirically measure the impact of different locations on the latency with the core of the network (previously presented in chapter 3) on the validators' stability and performance. To measure those, we collected two types of software and metrics. One relates to the publicly available data on the blockchain, while the second requires measuring the arrival time of messages and events to each tested node.

Scoring Ethereum CL's Duties

Shifting Ethereum's consensus mechanism to PoS unarguably increased the complexity of the protocol. As chapter 5 described, due to the economic incentive PoS defines for honest active validators, the profitability of the validators is directly linked to the steadiness and the correctness of their duties. Using the proposed methodology in section 5.3, the studies performed in this chapter required indexing and computing the performance of a validator using the stored data in the public ledger. We relied on the previously introduced open-sourced software *GotEth* in section 5.3.7 to index each of the duties that the control set of validators for the study performed over the defined period of epochs. The tool provides detailed data on which attestation flags were missed, the balance difference between epochs, a clear division of what duty generated which share of the reward, and even the MER each could have obtained per epoch. Thus, all these metrics become an ideal and homogeneous methodology for comparing the different validators.

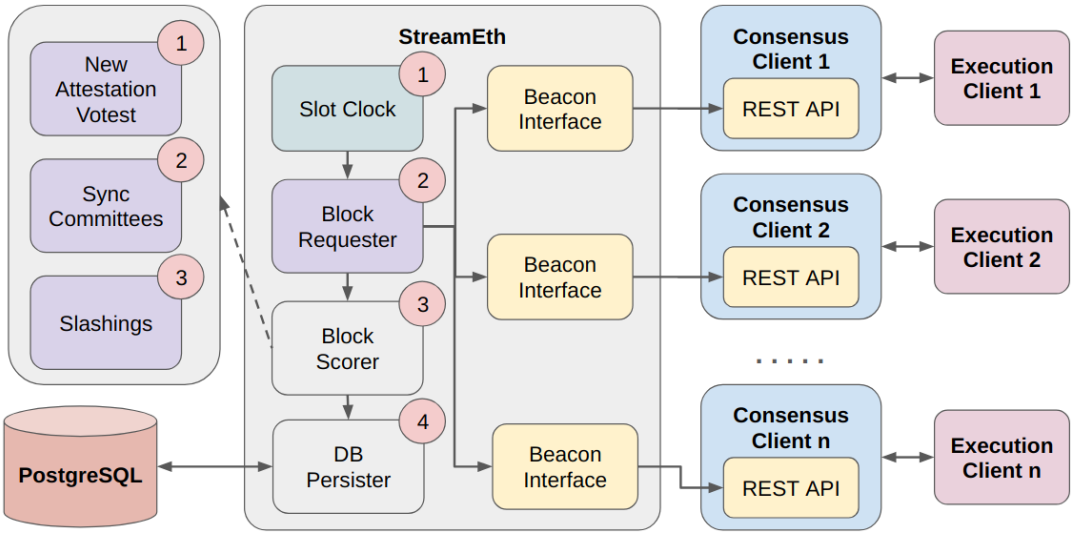

FIGURE 6.1: Internal diagram of the Ethereum's block scorer tool. In green, the internal structure of the tool showcasing the five main steps: 1) identify when each epoch and slot starts, 2) download the epoch duties, the beacon block on each slot (if it exists), the state at the end of the epoch, and the execution receipts, 3) with the raw chain data, compose the participation of each validator and the reward that each one got, 4) based on the chain state, compose the MER of each validator, and 5) persist of the metrics into the database.

StreamEth: a Consensus Beacon Nodes Monitorer We've already presented that not all the events occurring on a public network like Ethereum are publicly available. Some of this temporary data, like the arrival time of new blocks, the list of transactions available at the "Mem-pool", or the aggregated attestations locally available to compose a new block.

Such temporary data requires a real-time monitoring system to track and persist all the related metrics. In the context of identifying any existing differences originating from the extra delay in the location of nodes in different regions, we developed a second open-sourced tool called *StreamEth* [127]. The tool is based on the beacon node multiplexer *Vouch* [8], which allows the connection of a single validator client to multiple beacon nodes to provide some resilience at the development level.

StreamEth shares that same idea. However, it has a higher orientation towards monitoring, analysing, and persisting time-related events from each connected node. The tool can remain connected to the REST API of any number of beacon nodes, as figure 6.1 shows. It has an internal slot clock, which, based on the Genesis time of the network that it is configured for, can trigger the creation of a beacon block through the REST API right at the beginning of the same one (check figure 2.5 as reference). While the request has been generated and the tool waits for the beacon block reply from each connected node, it measures the time it takes each node to reply.

On the third phase of the operation flow in the diagram displayed in figure 6.1, the tool implements a specification-compliant block scoring algorithm. This block scoring aggregates the number of the included new votes, sync aggregates, attester slashing, and proposer slashing, generating a synthetic scoring system that later on will be used to compare them. The score is derived directly from the beacon chain rewards formulas presented in section 5.3.4, removing the *BaseReward* from the equation to make the score calculation faster.

As the idea is to measure the impact of the delay on the performance of validators

TABLE 6.1: Table with the versions used during the studies.

Short	Client	Version
LH	Lighthouse	v3.1.2-01e84b7
L	Lodestar	v1.2.1/5813d39
N	Nimbus	v22.9.1-ad7541-stateofus
P	Prysm	v3.1.1
T	Teku	22.9.1

and nodes, the tool subscribes to some streaming endpoints of the node's API. This way, the tool can track and timestamp every time it gets notified when a new block message is received. This allows us to compare the arrival time of messages such as new blocks.

Finally, as the tool's idea is to gather all those metrics for a later study of the same ones, the tool persists all these metrics from each node and the live network into a PostgreSQL database.

Deployment of the nodes

The study presented in this chapter is divided into two experiments. The first one is oriented to the impact of the profitability of validators based on their attestation capabilities, while the second one is related to measuring the effect that latency has on block arrivals and the capacity of nodes to compose better blocks.

We generated control groups of the five open-source clients for both studies to compare the locations and the clients. The list of clients and their respective versions are summarized in Table 6.1. Furthermore, as pairing each consensus client with an EL client became mandatory after the merge, we spawned the five CL clients with a matching Nethermind [133] instance.

Deployment for the Validators' Rewards Experiment Most cloud service providers cannot offer the same hardware resources in all the regions we wanted to test. Therefore, we used multiple cloud providers with different capabilities. To ensure that all clients had roughly the same hardware resources and due to the noticeable hardware limitations outside the EU and the US, we had to break the client deployment of locations such as Sydney, Singapore, and Toronto into two different machines (check table 6.2).

For this first study, since activating a validator in the mainnet requires a deposit of 32 ETH, we relied on Ethereum's Goerli testnet to activate 3.000 validators with the help of the Ethereum Foundation (EF) [60]. The division of these 3.000 validators was evenly distributed across the six locations, giving a total of 100 validators per client per location.

Deployment for the Block Scoring Study We experienced similar problems to the ones deploying the validator rewards study; fitting five CL clients with their respective five Nethermind clients under the same or similar machines was unmanageable. Thus, the clients and the machines were organized and deployed as table 6.3 summarizes. The data displayed and analyzed in the following sections belongs to the range of slots 5.760.722 to 5.888.722, which belongs to the range from February 9th, 2023, to February 27th, 2023.

TABLE 6.2: Hardware setup per location for the rewards study.

City	Hardware	CPU	Memory	Storage	Clients
Frankfurt	Baremetal	16c.	64GB	1.9TB NVMe	All
London	Baremetal	16c.	64GB	1.9TB NVMe	All
Warsaw	Baremetal	16c.	64GB	1.9TB NVMe	All
Sydney1	VM	16c.	32GB	850GB SSD	LH,L,P
Sydney2	VM	8c.	16GB	700GB SSD	N,T
Singapore1	VM	16c.	16GB	900GB SSD	LH,L,P
Singapore2	VM	4c.	16GB	700GB SSD	N,T
Toronto1	VM	16c.	32GB	910GB SSD	LH,L,P
Toronto2	VM	4c.	16GB	660GB SSD	N,T

TABLE 6.3: Hardware type on each tested location of the block score study.

City	Hardware	CPU	Memory	Disk	Clients
Helsinki	Baremetal	32c.	128GB	10TB NVMe	All
London	VM	32c.	128GB	7TB SSD	All
Warsaw	VM	32c.	128GB	7TB SSD	All
Sydney1	VM	4c.	32GB	2.5TB NVMe	LH,L,P
Sydney2	VM	4c.	32GB	1TB NVMe	N
Sydney3	VM	4c.	32GB	1TB NVMe	T

6.1.4 Validator Consensus Performance

The study of the validator's performance based on the geographical location starts by measuring the correctness of the attestation flags, as these ones significantly impact validators' rewards. To investigate why a single location would achieve fewer rewards than the rest, figure 6.2 shows the ratio of missed flags aggregated by location.

The current PoS consensus mechanism drastically changed the reward system in Ethereum. The protocol prioritizes rewarding validators' stability and continuous duty compliance. The reward retrieved through attestations represents the 61% of the gross reward a single validator can achieve. Thus, this first experiment compares the stability of performing attestations. To replicate the study and measure the impact of running clients in what we consider regions with more significant latency, table 6.2 includes the configuration of nodes we designed to perform the study.

Reward Comparison Based on Goerli Validators

The first part of the study compares the aggregated validator rewards per location between epochs 157.835 and 158.835, or in a human-readable format, between dates February 23rd 2023, and February 27th 2023. Intending to discover any hints of a possible under-performance of any specific region, figure 6.3 shows the achieved reward by the aggregated validators per location out of their respective MER.

Considering the aggregation of the validators per location as our validator control pools, we observe that most nodes achieved a similar reward when comparing it with their MER. It is expected to see drops in the achieved rewards when we compare validators in testnets with validators in mainnet. In this case, the Goerli testnet is publicly open for participants to collaborate without needing fiat collateral to ensure their participation. Thus, there is a higher ratio of participating nodes that are

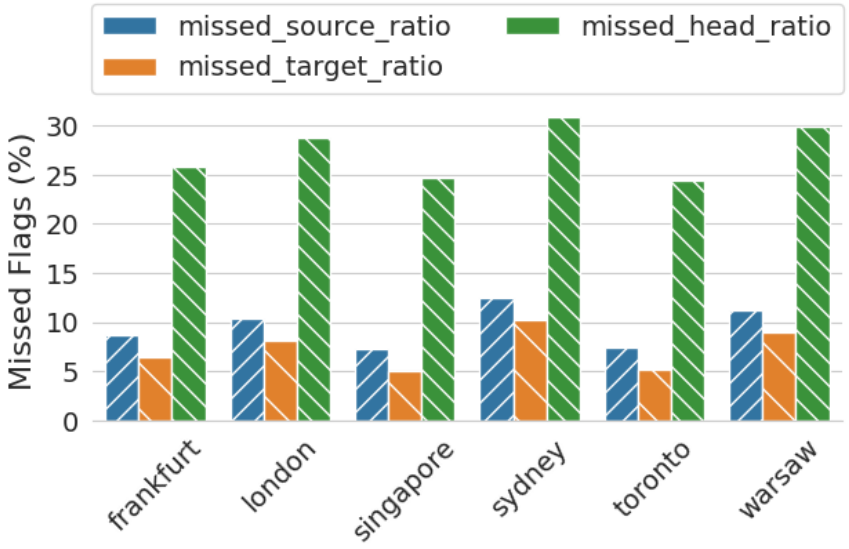

FIGURE 6.2: Average missed attestation flags.

not properly maintained, wrongly configured, or directly disconnected. This ultimately impacts the stability of the network, experiencing more missed blocks, more reorganizations, more missed flags, and thus bigger inclusion delays, lowering the MER compared with validators in mainnet.

With all these said, in figure 6.3, we still find a slight variation around an average reward of 80.2% for most locations. The most significant exceptions are recorded in Sydney and Warsaw, which fall to 74.3% and 76.8%, respectively. As explained before, the instance deployment differs in some locations. Nodes located in Frankfurt, London, and Warsaw share a similar infrastructure setup, in which the achieved reward is similar, varying between 76% and 82%. Nodes located in Singapore, Sydney, and Toronto also share a similar infrastructure setup between each other. However, Sydney achieved 10% less MER than the other two locations (84%). Since Sydney is the most remote location of the chosen setup, as most nodes concentrate in Europe and North America, a message is expected to have a slightly higher latency to reach nodes in such remote locations. Thus, this could show that network latency significantly impacts performance. It is remarkable that validators hosted in nodes with apparently "worst" connections, i.e., nodes in Singapore, extract more rewards than "better" connected ones, such as nodes in Frankfurt. This indicates that hardware also plays a vital role in achieving a more significant share of the achievable rewards.

Missed Heads The head flag inside the attestation points to the head slot in the canonical chain. To send a correct head attestation, the validator must point to the head root and provide this attestation with an inclusion delay of 1 block. Otherwise, the head attestation is given as wrong. This explains why it is the most commonly failed flag among validators in the network. Failing the head attestation flag could mean that the node falls more into reorgs or that the attestation is not sent in time and, thus, not included in the next block (inclusion delay=1). Figure 6.2 shows the average flag failure per location, where we can see that the average head attestation flags' failure rate is 27.4%. The figure shows that nodes in London, Sydney, and Warsaw failed over that average 28.7%, 29.9%, and 30.8% missed head flags, respectively.

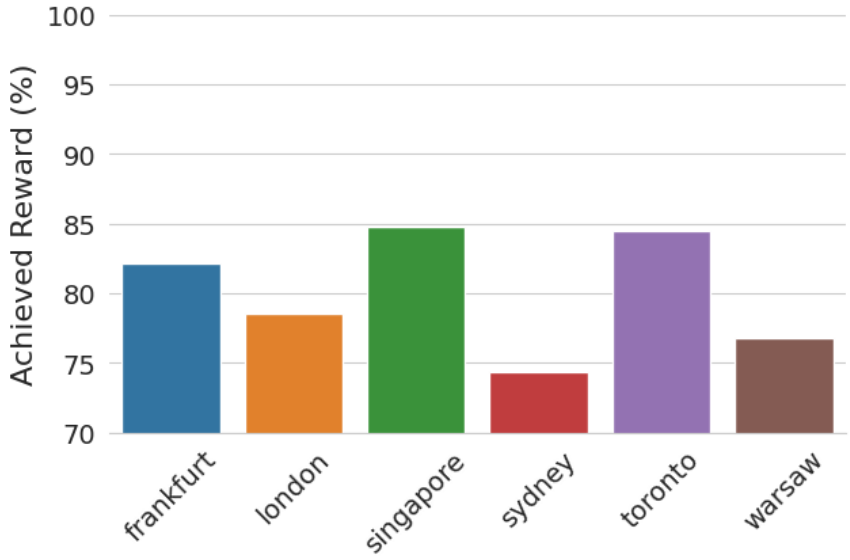

FIGURE 6.3: Average achieved reward per location.

Missed Targets Conversely, the target attestation flag is the least failed flag and the one that brings the most rewards out of all three flags. This is why we define that if the target flag is failed, the node is most likely out of sync and has been like this for a while. Figure 6.2 shows a similar pattern for the target flags, with nodes in Frankfurt, Toronto, and Singapore missing them 5% and 6.3% of the time. On the other hand, London, Warsaw, and Sydney nodes stay beyond that average, reaching the missed ratio of 8% 8.9% and 10.2%, respectively. It is clear, then, by analyzing the aggregation between the number of missed head and target flags, that despite Sydney sharing the same hardware with Toronto and Singapore, it falls out of sync almost double the times.

Proposer Duties Validators also earn rewards from proposing blocks when they are randomly chosen. Block rewards are sporadic but very high, so it is frustrating for validator owners to miss the chance of gaining such a high reward with a straightforward duty. When proposing a new block, the node must be fully synced with the network and follow the chain head without delays. Not doing so could cause the validator to miss the block proposal (or do it very late), missing out on the substantial block rewards it generates.

As block proposers are randomly chosen at every epoch, figure 6.4 shows the aggregated ratio of missed proposer duties between locations. The figure shows that pool nodes on each location failed an average of 1.66% of block proposal duties, with Sydney nodes failing up to 4% of the block proposal duties. At the same time, Frankfurt, Toronto, and Singapore didn't miss any of the proposals. Once again, it is most likely that Sydney nodes tend to fall more into the out-of-sync state and, therefore, can not perform their duties in time.

Chain Reorganizations on Clients Nodes have different behaviours depending on the hardware they are running on and the location where they are placed. However, validators' achieved rewards or the ratio of missed duties are not the only methods to measure the performance of a node. When attesting, the high ratio of missed target flags is the first indicator of a stability problem on a particular node, or in

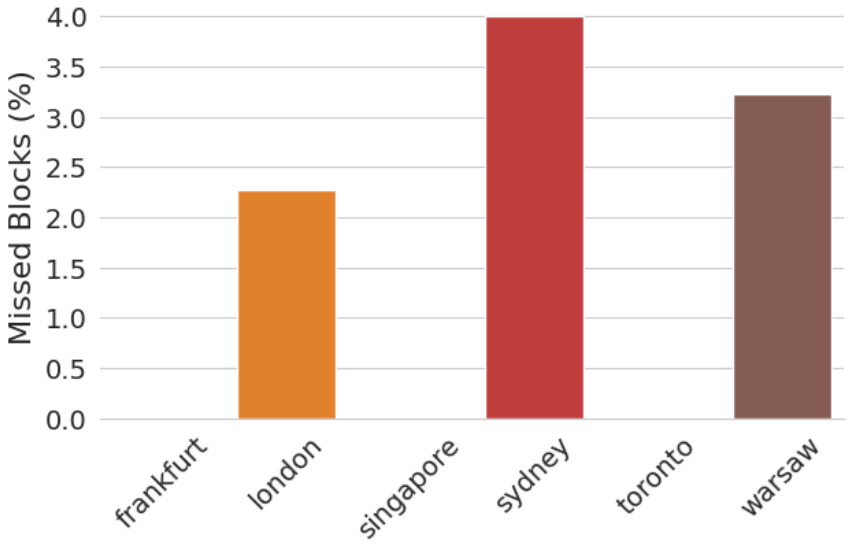

FIGURE 6.4: Aggregated missed blocks per location.

this case, in a location. It is hard for a validator to perform its duties correctly when the underneath node is not fully synced with the chain. Thus, we can interpret that nodes in Sydney, as a representation of less well-connected nodes, tend to lose synchronization more often. We have chosen reorgs (chain reorganizations) as a way of measuring the stability of a node. A chain reorg represents having to drop several blocks (with their states) and sync the canonical version of them because the node was in a non-valid chain variation. As the late arrival of messages can generate those minor forks, figure 6.5 shows the aggregation of reorgs registered in each location. The graph shows that Sydney has the biggest reorg average of the six different places, with eight more registered events than the average of 102 for all the locations. Without being an extremely large number, we can attribute the differences between duties accomplishment to the fact that the reorgs can also be defined by the number of blocks you had to drop and resynchronize. Unfortunately, the node does not offer us this information. We will discuss this step in detail in the following section.

6.1.5 Block Scores

With the significant differences we spotted between the arrival times of the different instances deployed in mainnet (the most stable network in the Ethereum ecosystem), we wanted to study the impact of this arrival latency on the clients' capabilities to compose blocks.

Following in order the rewards quantities that a validator received over time in the Consensus Layer, the resulting rewards of block proposals follow attestations with a $7,6\%$ of the total reward. Thus, with a synthetic block score (check section 6.1.3 for a more detailed overview), we decided to benchmark the differences that each client and each location produce.

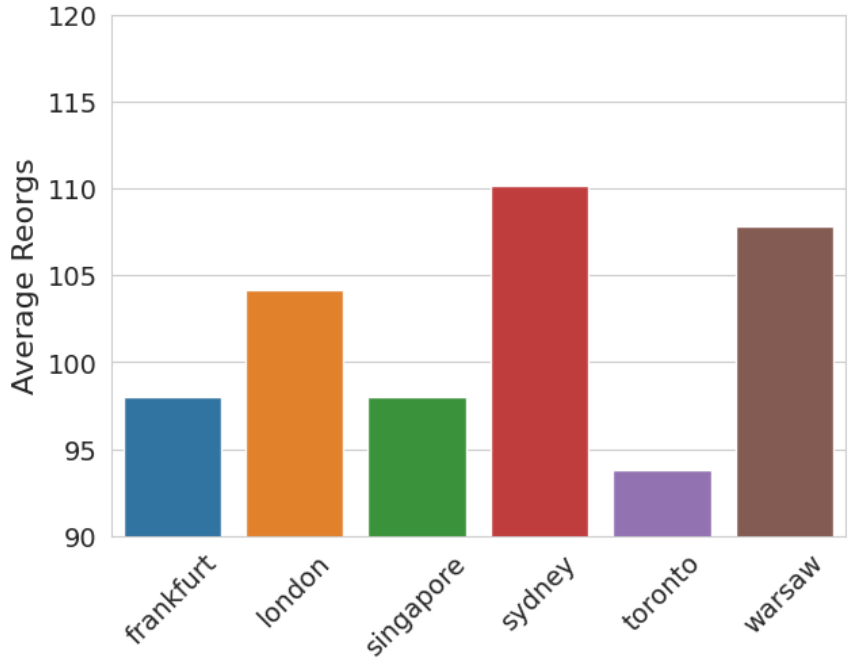

FIGURE 6.5: Number of chain reorganizations per location.

Beacon Block Generation Differences

Figure 6.6 shows the aggregated score of each block across the clients of each location, where we appreciate insignificant differences. Once again, there is a clear dominance of nodes located in Helsinki, outstanding over nodes in Sydney, London, and Warsaw that achieved 1.37%, 4.29%, and 5.16% less score, respectively. Although each of the locations should have enough time to get the same attestation aggregations in the course of the last 4 seconds of a slot, different factors could produce this difference in the average score of the blocks:

Message Propagation Latency Higher latencies when receiving messages clearly limit the number of aggregations that you can include in a block. Figure 6.7 displays the Cumulative Distribution Function (CDF) of the message arrival time in each location, where the Y axis represents the normalized percentiles between ranges 0 and 1, and the X axis the arrival time in seconds. We can read the figure as the 50th percentile of blocks (0.5 on the Y axis) in Helsinki arrived in 1.44 seconds or less. We can see that, despite London having the second-best median of arrival times, 2.18 seconds, it has one of the worst 90th percentiles of 4.15 seconds. The large tail of block arrivals beyond 4 seconds represents 10% of the total tracked messages and it partially explains the block score differences between locations.

The Aggregation of More New Votes in a Block Receiving messages later means adding fewer votes in a new block, which are the ones producing the rewards for the block proposer. Figure 6.8 displays the average number of new votes included at each location, including the average of their correctness. The figure clearly shows the dominance of nodes in Helsinki that not only aggregate more new votes but also have, on average, more correct attestation flags.

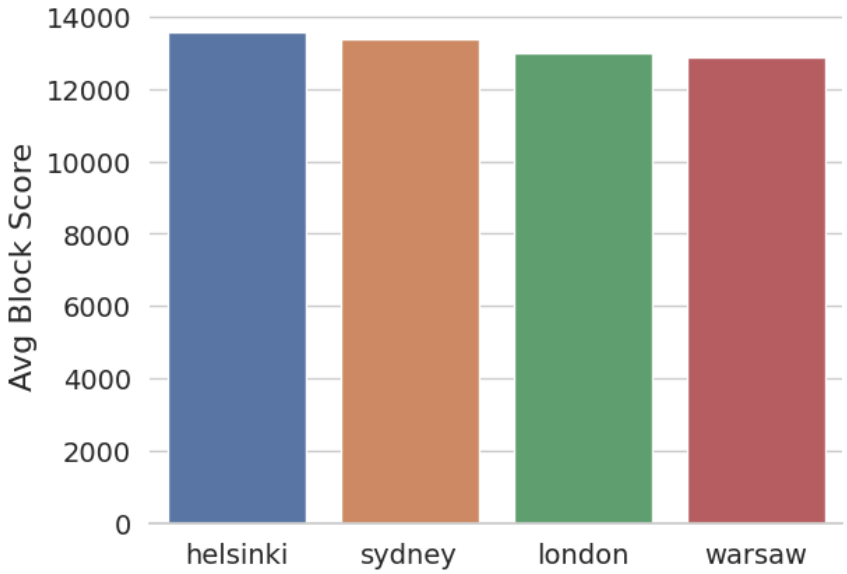

FIGURE 6.6: Average block score per location.

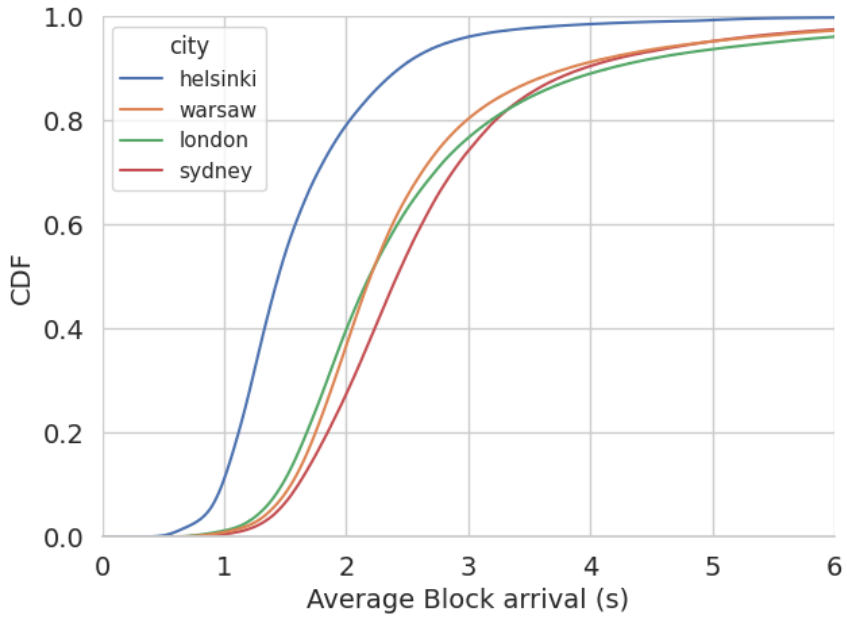

FIGURE 6.7: Average arrival latency of block messages inside slot time per location.

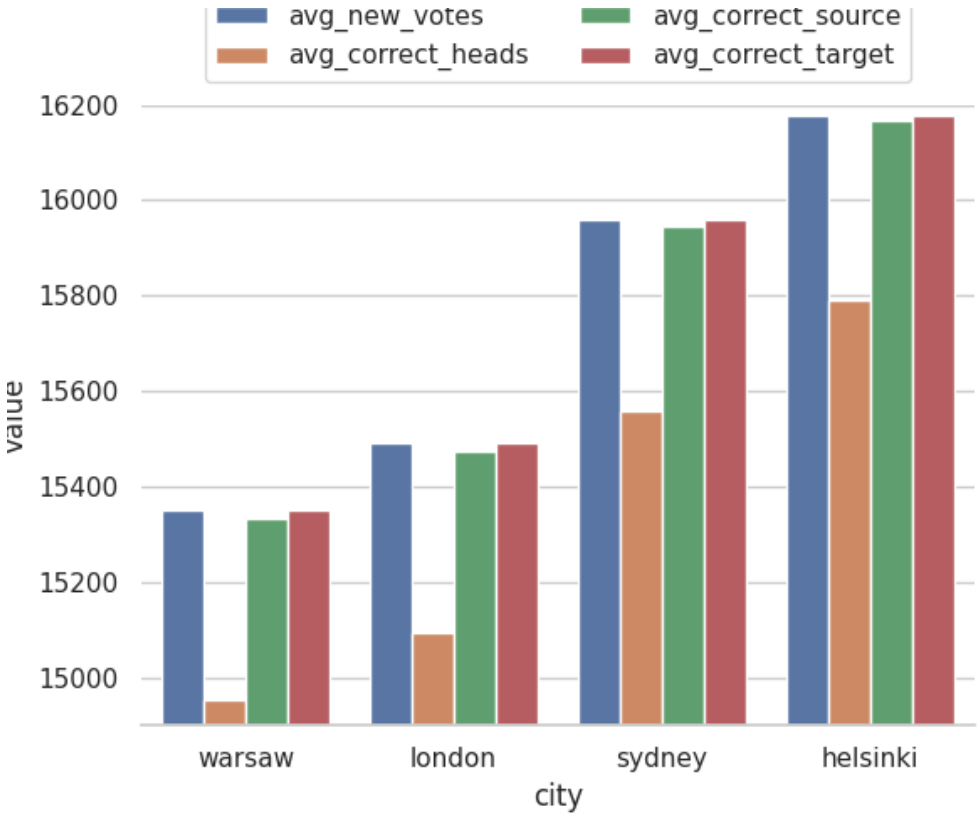

FIGURE 6.8: Number of new votes and their correct flags on each location.

Desynchronization of Beacon Nodes Desynchronization is, even if we put endless effort into avoiding it, one of the major drawbacks we found to explain the difference in average score between locations. Of course, a beacon node can not process new messages and generate new blocks if it gets out of sync with the head of the chain. Several events could cause this to happen, such as big local reorgs caused by higher latencies or by hardware limitations (slow disk where to prune states or insufficient CPU available to validate messages on time). Figure 6.9 displays the percentage of slots each node was down versus the number of slots we measured. In the picture, we can appreciate that Helsinki was barely unsynchronized. We can also appreciate how Lodestar in London affected the previous averages, with a 37,74% of the measured slots out of sync. The results are more puzzling when we compare it with Warsaw, concluding that despite having similar hardware, it was stable and in harmony with the rest of the clients at around 5,25% downtime, and despite this, it performs worse than London both in correct flags and block score.

Latencies on Block Arrival We have already introduced the importance of message broadcasting latencies within the slot time range. Shorter notice of messages such as blocks or aggregations might be critical to ensure that validator duties are correctly achieved. As we would imagine, different locations with different connections with the rest of the network could generate different network perspectives. This way, if the defined 4 seconds to distribute a message weren't enough, we could expect that those regions with higher latencies would perform poorer than the better-connected ones. To corroborate our hypothesis, we tracked the arrival of block messages to each client in the four different mainnet instances. The tool would

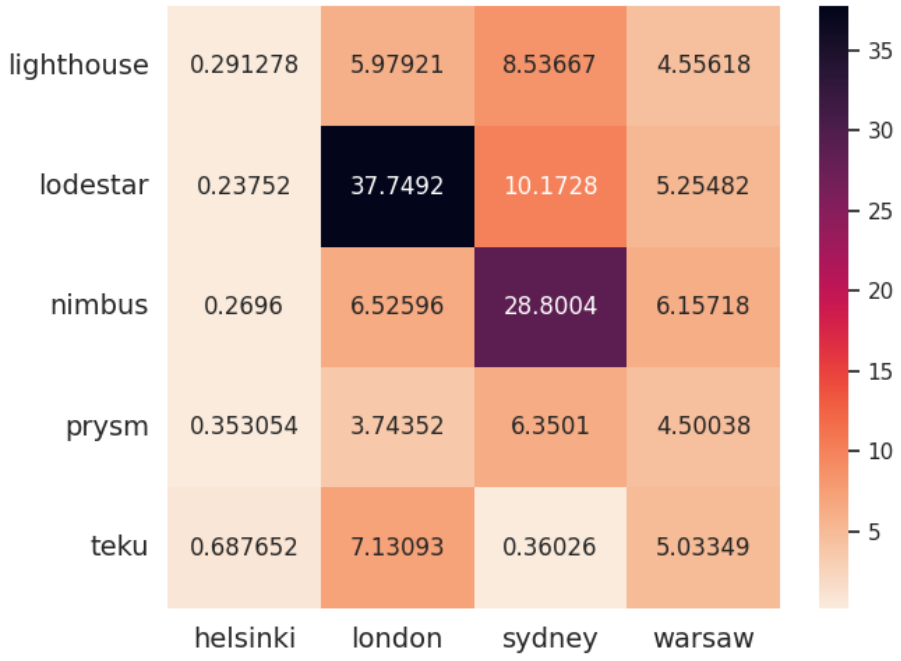

FIGURE 6.9: Percentage of slots each beacon node was out of sync in each location.

first subscribe to the beacon node's API to stream the arrival of new blocks, indexing the event with the notification timestamp. From the difference between the local timestamp and the time the slot started (second "0" of the slot, when block proposers should publish the block), we have aggregated the arrival times, distinguishing the distribution from the following figure.

Figure 6.10 shows how clients in Helsinki received the block messages significantly sooner. We would expect from previous experiences that the European area has better connectivity in general terms, and this graph proves it: on average, nodes in Helsinki received blocks 1.06 seconds sooner than nodes in Sydney. However, even though we are not making any distinction across clients (all clients were aggregated) to have a fair comparison between the locations, the differences between London, Sydney, and Warsaw are not as remarkable as the arrival times in Helsinki. With an average block arrival time of 2.60, 2.68, and 2.53 for each, the figure shows that the major differences between the instances could originate from the distinct machines chosen per location.

Although we tried to have the most similar machines in each location, Cloud Service providers couldn't offer the same hardware tier across the four locations. With a clearer dominance of resources from the machine in Helsinki, even though it had to share the disk among ten different clients (five CL + five EL), it still had a powerful bare metal machine with 32 CPU cores and 128GB of memory. Limited CPU resources could become a bottleneck when a client receives and validates a block message, as they generate computational spikes every 4 seconds. Thus, to some degree, this CPU bottleneck can increase the measured latency of processing blocks, including the time our tool got notified. Of course, different clients mean different implementations. The heat map in figure 6.10 breaks down the block arrival times per client and location, showing some differences across the clients. Without being highly dispersed among the clients, it is clear that some clients received and processed the block faster than others during our study. Despite showing the same

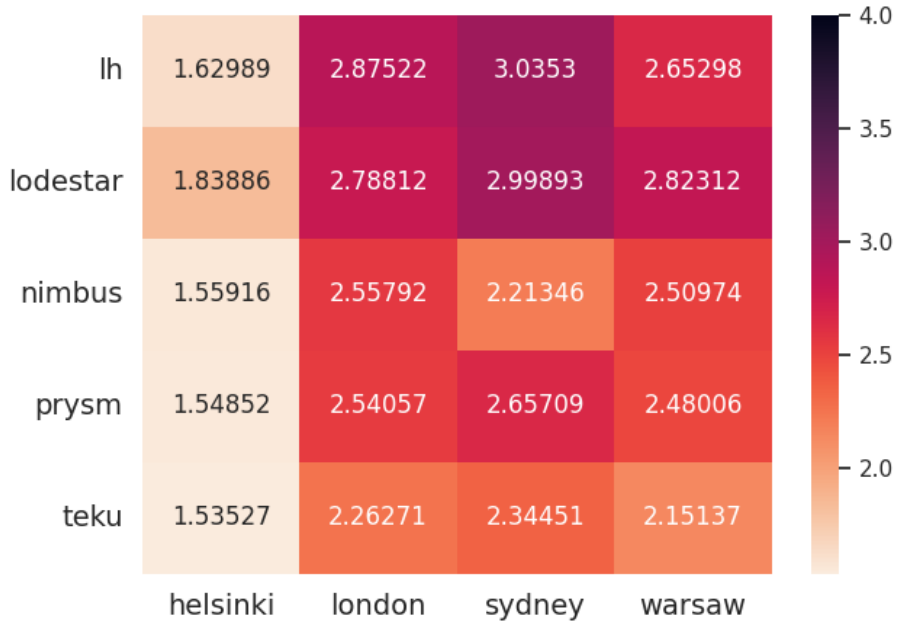

FIGURE 6.10: Average block arrival time per client and location.

distribution as the previous figure 6.9 among the different locations, Lodestar is in the order of 300ms slower in receiving and processing new blocks.

There are many reasons that could cause a later message arrival. On the one hand, we have the number of connected peers, where Lodestar and Prysm stay at 50 to 55 number of simultaneously connected peers, Lighthouse and Teku stay with an average of 80 to 100 connections, and Nimbus out-stands the rest with 160 direct peers. However, on the other hand, clients must allocate more computational power at the arrival of messages to process and validate the messages or to update the local beacon state. For this reason, the bigger the number of simultaneous connections the node has, the more messages you need to process. Thus, this typically generates CPU usage peaks every 4 seconds, and as it happens with higher latencies, both directly impact the client's performance.

6.1.6 Conclusion

This section of the chapter presents a distinct methodology that can be used to analyze the impact of latency on the performance of Ethereum CL nodes located around different geographical locations. In the presented results, we have identified that Ethereum CL's default 4 seconds for broadcasting gives enough margin to propagate and receive all the necessary messages (i.e., sync committee attestations and block proposals) in most geographical locations, at least if the clients are running in instances with minimum hardware specifications.

We have empirically demonstrated that some regions, such as Oceania and Southeast Asia, have higher latency distributions when receiving block messages for the first time; with some locations receiving 10% of the messages beyond the 4 second mark. This makes nodes more likely to lose the correct head of the chain, leading more often to reorg their local chain and ultimately failing or performing duties more poorly because the node is not fully operational. This gets mirrored when comparing the rewards achieved between the available locations, where Sydney nodes earned 10% fewer rewards from the MER than the rest. We have demonstrated that

even though the hardware requirements to participate in a Post-Merge Ethereum are substantially smaller than its predecessor PoW consensus, there are some minimum requirements to run both EL and CL clients without a hardware bottleneck.

Furthermore, the section compares the networking performance of the different available Ethereum CL clients. The study presents that under optimal networking and hardware conditions (i.e., the instance in Helsinki), there are barely any differences between clients, i.e., a similar downtime or out-of-sync time. We have identified that hardware limitations directly increase the latency (mostly from message validation). Thus, nodes face more downtime as they might get out of sync, or could potentially add fewer new votes to their blocks.

We can conclude the section by stating that there is indeed a performance impact related to the connectivity of a node. Still, it is only significant when the hardware is not properly dimensioned. With the presented clear difference in block arrival latencies across locations, nodes further away from the core of the network can see their stability and performance reduced. Reaching even critical stability problems if the hardware is not slightly over-dimensioned. In future work, we aim to explore a real-time model able to aggregate and compare all the presented parameters to monitor and alert the performance of a client with or without a validator.

6.2 DVT in Ethereum's PoS: Gains and Loses

6.2.1 introduction

Resilience is vital for any application that prevents service outrages from failure in critical infrastructure [162]. With multiple available methods to achieve it [40], resilience has been present in the modern era since humans were aware of the system's likelihood of failure [197], offering a backup plan to overcome the disruption.

As systems tend to grow in size and complexity, so does the error and risk rate [120]. The concept of "resilience" also reached the software engineering field, which greatly benefited from it as single-core hardware and programs evolved to bigger and more complex multi-threaded or even distributed clusters of nodes at High-Performance Computing (HPC) [53]. Then, the idea of "fault tolerance" was introduced, aiming to provide solvency to systems, preventing them from critical failures [156] or at least define the recovery process to do so.

Following the same philosophy, Ethereum's Proof of Stake (PoS) includes several resilience techniques that ensure the protocol can tolerate certain failure rates or recover from a vital disruption. The consensus is designed to finalize beacon states[1] as long as the network achieves a 66% of honest participation on the consensus by attesting the corresponding blocks [7].

On a separate level of resilience, at the software level, the Ethereum community supported five teams to develop and maintain five open-source clients to participate in the network. This variety of software reduces the chances of a "single point of failure" at consensus from a software level. At the network level, based on the distributed nature of the project, each node relies on a set of nodes per GossipbSub topic [199] to receive chain notifications such as blocks, attestations, or sync committees. A rotation of these connections, with a fast node discovery and the aggregation of more advanced techniques like message gossiping, considerably reduces network attack vectors like Eclipse or Sybil attacks.

[1] Beacon State refers to the global state of the entire network, including the status and balances of all validators, while providing a snapshot of the consensus among validators.

We've seen how the network considers several layers of resilience at the consensus level, at the software level, and at the network layer. However, it doesn't provide any resilience that could be applied at the validator or deployment level. In fact, the consensus somehow prevents such techniques, as sharing different duties at the same slot signed with the same key leads to an imperative penalty, a "slashing" [77]. This consensus design showcases the thin action line that validator agents work in, where their retribution for participating in consensus might stop if anything happens to the machine hosting the software. Some projects like Vouch [8] have emerged, allowing the reduction of this single client failure. In this case, Vouch is a multiplexer client placed as a middleware between the validator and the beacon node. Based on different strategies, it decides which node is chosen to participate in the network, either the fastest replier node or the node that produces the highest reward. This schema guarantees that as long as one of the nodes in the cluster stays up, the validator will still be able to submit its duties. However, as the penalty reports a significant risk for complex developments that ensure a 100% uptime, there is still a huge gap for a validator multiplexer with high-security guarantees.

It is at this step that new technologies like Distributed Validator Technology (DVT) have emerged to fill the missing chapters in distributed staking software. Based on the idea of signature aggregations using private BLS key shares [22] from a Distributed Key Generation (DKG) protocol/ceremony [74], this technology provides a specific degree of fault tolerance based on the number of participants in the cluster. The bigger the cluster, the higher the fault-tolerance degree in terms of nodes that can still produce the validator duties.

Despite the idea sounding good on paper, there is a side effect problem with this DVT technology: it adds an extra step before validators broadcast their duties to the network, which is the aggregation of partial signatures. Gasper's fork choice [149] contemplates the idea of asynchronously arriving commits. However, to prevent some attack vectors, the PoS protocol of Ethereum sets economic incentives to avoid them. The protocol sets limited time ranges to compose and broadcast each of the duties, ensuring that the duties need to be pseudo-synchronous for an optimal reward.

In other words, to simplify and ease the chain's fork choice and consensus, Ethereum's PoS reduces the consensus rewards as validators take longer to accomplish and broadcast their duties. While DVT offers a novel, more resilient way of staking, it is unclear whether it can match the performance of classic (non-distributed) validators. This opens a legitimate question of how this tight time sensitivity on the protocol side can interfere with the profitability of a DVT validator cluster. In particular, for setups that include clusters of nodes distributed worldwide and from different cloud providers, it is necessary to demonstrate that despite their latency, DVT can still provide the same level of performance as classic validators. This is the objective of this study.

In this section, we present a thorough analysis of the performance and profitability of DVT validator clusters (the approach of Obol Labs, to be precise) compared to the classical validator approach. We present a clear comparison of the rewards obtained using both approaches, showcasing the latency restrictions that could affect them and detailed insights into the internal metrics of a DVT cluster.

6.2.2 Distributed Validator Technology: How Does It Work?

Distributed validator technology, or DVT, is a critical security primitive that allows a single Ethereum validator to be run on a cluster of nodes working together as a

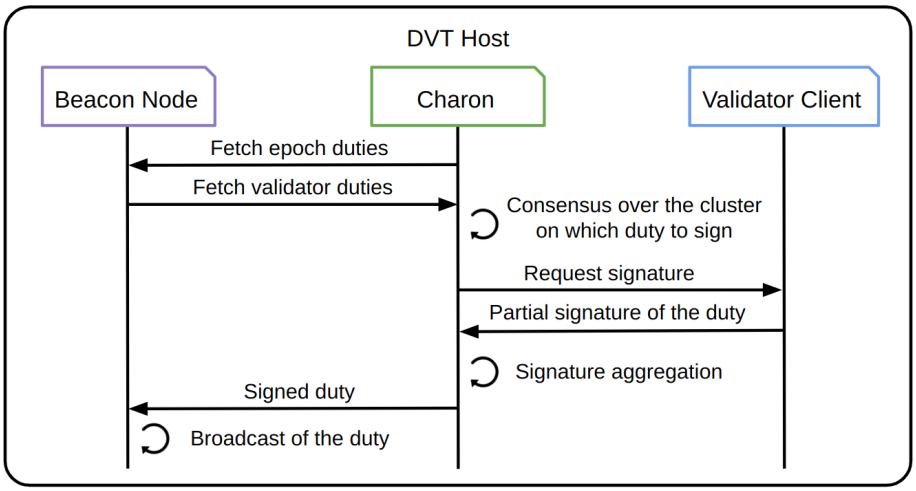

FIGURE 6.11: Internal communication between the beacon node, Charon, and the validator client when performing a consensus duty of a DVT validator.

distributed validator. The concept of "splitting" the validator into a cluster removes the single point of failure for validators, creating an active-active redundancy with a failure threshold. However, the network is agnostic to this sub-division and perceives the cluster as a single validator. DVT provides some safe margins where if one or several cluster nodes fail to send their partial signatures, the distributed validator can keep performing its duties as long as enough nodes (over the threshold) submit their partial duties.

In this section, we will closely examine the DVT approach of Obol Labs[2], monitoring and analyzing the performance of a few DVT validator clusters using Charon [32]. Charon is a middleware client that sits between the beacon client and the validator client of each node within a distributed validator cluster. It provides the logic for communicating with the beacon node and the cluster, scheduling and deciding what to sign and when to sign.

Obol's approach to DVT starts from a Distributed Key Generation (DKG) [94, 112, 176, 73] ceremony, where the cluster participants generate their private BLS [11] key shares and then sign the validator deposit. Once the deposit is processed, the cluster starts contributing to Ethereum's consensus in a collaborative effort. As each DVT cluster participant runs its Charon client, check figure 6.11 as a reference, which, in collaboration with the beacon node, tracks and schedules the duties that the given activated validator or validators need to accomplish. However, the cluster must still sync and decide what to sing before aggregating their partial signatures.

Obol relies on a private network between the users participating in the cluster. The network, besides being used to propagate the partial signatures, creates the space to play a Quorum Byzantine Fault Tolerance (QBFT) [1] consensus game to help the cluster agree on which duty the sign as a single validator. Thus, when a new duty is planned for a validator, the validator client retrieves the duty to be signed and sends it to the Charon client. The Charon client is then waiting for enough partial signatures from the rest of the nodes. After receiving enough partial signatures to meet the threshold, Charon broadcasts the signed duty to the beacon node, which broadcasts it to the network. The threshold of each cluster (minimum number of

[2]https://obol.tech/

partial signatures to perform duties) depends on the number of nodes in the cluster. A cluster can stay active as long as more than 66% of its participants send their partial signatures. For a more in-depth explanation of the DVT, please refer here.

6.2.3 Methodology

Comparing the consensus performance between DVT and canonical validators required multiple hardware and software resources. This section will summarize all the used requirements, providing a detailed description in case the experiment wants to be replicated with further versions as the DVT technology keeps polishing implementation details.

Software services

Participating on Ethereum's network in a post-merge scenario requires a wide set of software as introduced before with figure 2.3. Making the whole software setup a bit more complex, participating in an Obol DVT cluster implies relying on a fourth extra software, Charon. The whole client setup includes the following layers:

Execution Layer Nodes Since "The Merge" happened in September 2022, to follow up with the canonical state of the chain, the beacon node needs to validate the EVM interaction from users to accept or propose a new block. Thus, it needs to be paired one to one with an execution layer client. There are quite a few options to choose from; however, we relied on Nethermind[3] on its versions v1.17.1 and v1.17.3 for all our cluster deployments to support the client diversity in the network.

Beacon Nodes and Validator Clients On the consensus layer, as chapter 4 presented, the ecosystem has five main open-sourced software to keep track of the chain's head and track the fork choices of the consensus. However, as described in section 6.2.2, not all beacon nodes support the necessary endpoints at the REST API when writing this section. Obol's DVT client needs some specific endpoints:

- *POST/eth/v1/validator/beacon_committee_selections*
- *POST/eth/v1/validator/sync_committee_selections*

For this reason, since only two out of the five available clients had a reliable API, we relied on those two available options for the study: Lighthouse [4] on its versions v3.5.1 and v4.0.1, and Teku [5] on its versions v23.3.0 and v23.3.1.

To maintain consistency between the validator clients and the beacon nodes and ensure optimal interoperability, we decided to pair them with their respective validator clients on identical versions.

Charon As previously mentioned, deploying a cluster of DVT validators requires some extra complexity between the beacon chain's client and the duty signer validator client. In this study, we will be taking a closer look at Obol's implementation, measuring their DVT software, Charon, on its versions v0.14.0 and v0.15.0.

[3]https://github.com/NethermindEth/nethermind
[4]https://github.com/sigp/lighthouse
[5]https://github.com/ConsenSys/teku

Setup replication To ease the deployment of all these clients, ensuring the same configuration in all of them, we relied on a *docker-compose* setup [6] which is based on the official configuration provided by the Obol team [7].

This setup includes:

- Nethermind configuration for Teku and Lighthouse.
- Teku configuration (beacon node and validator client).
- Lighthouse configuration (beacon node and validator client).
- Charon as DVT client.
- Prometheus for metrics gathering.

6.2.4 Performance Indexes

Comparing the steadiness and profitability of a validator in Ethereum's consensus is not such an easy task; it involves many parameters that are hard to quantify and compare across multiple validators. To do so, we used a set of pre-defined and tested indexes that aggregate distinct metrics to help understand the performance of a node or a validator.

Maximum Extractable Reward (MER) We calculated both validator sets' rewards at a given epoch using the reward models proposed by chapter 5, the Maximum Extractable Reward (MER). The MER indicator assessed the maximum attestation and sync committee rewards a validator could get at each epoch if all the duties were performed correctly and within the optimal time window. The concept of MER allows for calculating the ratio of achieved rewards out of the MER. This allowed us to compare the profitability impact of running a distributed validator versus its canonical option.

Node Score To compare the distributed validator technology with its counterpart, a standard validator, we must quantify how much overhead each node of the DVT cluster adds to the overall performance of the cluster. Measuring a single client's uptime and steadiness is generally easy, as we could only track the signature aggregation time. However, this metric is not that easy to track on a distributed cluster of validators that need to share messages across multiple nodes. In this case, we adopted Obol's Beacon Node Score to measure the performance of each cluster node. This score represents the individual score of a single node from the rest of the cluster's perspective.

The score represents a synthetic parameter sensible to the error rate of the beacon node's REST API and to the latency between the rest of the nodes participating in the DVT cluster, which is summarized in the following equation:

$$Score = 0.5 * (1 - (10 * \frac{APIerrors}{APIlatency})) + 0.5 * (1 - max(perc99(P2pDVTlatency) \tag{6.1}$$

[6] https://github.com/tdahar/charon-distributed-validator-cluster
[7] https://github.com/ObolNetwork/charon-distributed-validator-cluster

Provider	CPU	Memory	Storage	I/O Speed
OVH	Intel Xeon E-2274G 8x	32GB	900GB	R: 422MiB/s W: 141MiB/s
Digital Ocean	Intel Xeon Plat. 8358 4x	32GB	600GB	R: 234 MiB/s W: 78.3 MiB/s
Hertzner	AMD Ryzen 9 5950X 32x	128GB	10.5TB	R: 512MiB/s W: 511MiB/s

TABLE 6.4: Hardware specifications of different cloud service providers.

Data Collection Tools

Generating a comparison of this magnitude requires a reliable generation and collection of all the involved metrics. With this purpose, we've selected a small variety of software that helps us identify the performance of the validators and monitor internal DVT operational metrics.

GotEth We employed the *GotEth* software previously presented at section 5.3.7 to index each controlled validator's duties accomplishment and their respective MER over the study time range.

Prometheus Most software nowadays produces internal metrics for debugging and monitoring purposes. Following the industry's tendency, the Ethereum ecosystem relies on such techniques to provide real-time information using top-leading tools such as Prometheus[8]. Prometheus is a time-based database service that captures all the metrics each tracked endpoint offers on a specific *HTTP* path. It allows the configuration of multiple targets, scraping their metrics at the desired frequency, which can be accessed later through evaluation and ruling expressions.

Node Cluster Deployment The experiment aimed to analyse the DVT performance over different cluster sizes, geographical locations and two cloud providers to give the experiment more robustness. Furthermore, this wide variety of hardware providers and software combinations helps stress-test the DVT cluster aggregation times for extreme cases. We decided to test three of the most relevant DVT clusters, the ones that add one extra node of threshold to the signature aggregations:

1. A **DVT cluster of four nodes** has a signature aggregation threshold of three nodes. The node deployment is displayed in table 6.5.
2. A **DVT cluster of seven nodes** has a signature aggregation threshold of five nodes. The node deployment is displayed in table 6.6.
3. A **DVT cluster of ten nodes** has a signature aggregation threshold of seven nodes. The node deployment is displayed in table 6.7.

For a fair comparison across the different node deployments, we've activated a total of 6.000 validators in Ethereum's Goerli testnet[9] (check Section 2.3.3 for further details about the network organization on Ethereum). 3.000 of those validators were

[8]https://github.com/prometheus/prometheus
[9]Entire list of validators is available at https://migalabs.io/reports/MigaLabs-Obol-Annexe.pdf

Provider	Location	Services (Beacon Node + Validator)	Node Name
OVH	Frankfurt	Lighthouse + Lighthouse	happy-body
OVH	Strasbourg	Teku + Teku	precious-food
OVH	Warsaw	Teku + Teku	expensive-mountain
Digital Ocean	London	Lighthouse + Lighthouse	mindful-movie

TABLE 6.5: Cloud provider hosting the four-node DVT cluster.

Provider	Location	Services (Beacon Node + Validator)	Node Name
OVH	Frankfurt	Teku + Teku	unusual-state
OVH	Strasbourg	Teku + Teku	selfish-rule
OVH	Warsaw	Teku + Teku	determined-party
Digital Ocean	Bangalore	Lighthouse + Lighthouse	enthusiastic-area
Digital Ocean	Frankfurt	Lighthouse + Lighthouse	affectionate-day
Digital Ocean	London	Lighthouse + Lighthouse	jolly-life
Digital Ocean	Singapore	Lighthouse + Lighthouse	cloudy-flowers

TABLE 6.6: Cloud provider hosting the seven-node DVT cluster.

Provider	Location	Services (Beacon Node + Validator)	Node Name
Digital Ocean	Bangalore	Lighthouse + Lighthouse	dangerous-mobile
Digital Ocean	Toronto	Lighthouse + Lighthouse	clear-fish
Digital Ocean	Frankfurt	Teku + Teku	beautiful-word
Digital Ocean	London	Lighthouse + Lighthouse	amazing-tea
Digital Ocean	Singapore	Lighthouse + Lighthouse	plain-news
Digital Ocean	New York	Lighthouse + Lighthouse	tough-city
OVH	Beauharnois	Teku + Teku	fine-adult
OVH	Frankfurt	Teku + Teku	alert-waterfall
OVH	Strasbourg	Teku + Teku	delightful-dates
OVH	Warsaw	Teku + Teku	powerful-road

TABLE 6.7: Cloud provider hosting the ten-node DVT cluster.

split evenly, first in sets of 1.000 validators, one for each of the described DVT clusters, and second, each validator was split in the respective private key-shares for the cluster size.

6.2.5 DVT Cluster VS Canonical Validator Setup Comparison

To evaluate Obol DVT, we performed a long multi-phased experiment that lasted for 10.000 epochs, about a month and a half of continuous execution. The purpose of running for such an extended period was to provide robust statistical analysis, as a shorter time range could subject the measurements to network perturbations, non-finality periods, or bias the profitability of a set of validators based on being "lucky" more often and provide an unusual number of blocks. The experiment was performed on Ethereum's Goerli/Prater network, and it was run between the epochs 163000 (Mar-18-2023 00:40:00 UTC) and 173000 (May-01-2023 11:20:00 UTC). The Ethereum Prater network is a testnet, and as such, it may have dangling validators that may not be running. This means there are more missed blocks than in the Ethereum Mainnet network, resulting in delayed attestation inclusion and more chain reorgs. These conditions are harder than Mainnet, which stresses the software even further. The experiment ran three clusters described in tables 6.5 6.6 and 6.7 with 1.000 validators attached to each. Therefore, the experiment involved running a total of 3.000 distributed validators. The following sections present the results of comparing the performance of the DVT validators with the canonical validator setup counterparts.

Steadyness of the Validator

In such complex protocols, software performance depends on multiple conditions: hardware resources, network connectivity, software optimizations, etc. However, in Ethereum's case, specifically on its PoS version, this performance can be reflected in two main aspects: the accomplishment of validators' duties and the timing and correctness. The following sections will analyse these aspects individually, comparing the results of the different instances deployed worldwide.

The coming-up analysis was performed between epochs 163.000 (Mar-18-2023 00:40:00 UTC) and 173.000 (May-01-2023 11:20:00 UTC) on Ethereum's Goerli/Prater network. As significant upgrades were performed on Charon's software, we upgraded the versions of all software (EL client, Beacon node, Charon, Validator client) at the middle of the run in epoch 168511 (April-11-2023 14:30:24 UTC). This system setup upgrade enabled the software's performance progression to be benchmarked, specifically Charon's DVT implementation.

Missed Attestations At every epoch, validators are assigned to attest to one of the 32 slots, having the obligation of always submitting one attestation per epoch. These attestations, as section 2.3.1 introduced, create the link across blocks, enabling the chain to finalize previous states. To do so, each validator needs to provide three flags: one that links to the last epoch (source), one for the current epoch (target), and a third one that references the last available block (head).

A wide range of things could go wrong with the attestation's flags. Due to the sensitivity of the head slot, it requires having the same perspective as the majority honest part of the validators, and it would only be counted as correct if the attestation with the correct head flag has an inclusion delay of 1 slot (it must be included on the right next proposed block). This means that there could be three major reasons

to miss the flag: i) the beacon node had a wrong vision of the chain, i.e., the block on that slot somehow arrived late to the beacon node, causing a fork from the canonical chain, ii) the there was a missed block right after our attesting slot; thus, the attestation couldn't be registered to the beacon chain on time, and iii) the attestation was broadcasted late to the network, and it wasn't included in the next block. The many reasons for missing the head flag make it the most missed flag in an attestation. Especially in testing networks like Goerli/Prater, which generally have lower participation.

On the other hand, the source and the target flags are harder to miss. The wrong perception of the previous and the current epoch could only mean that the beacon node was unavailable or that the head of the chain was lost. For this reason, these two flags generally fail together and represent the steadiness of a validator. However, there is a slight difference between them: the source flag has a maximum inclusion delay of five slots, which is generally the reason why it is missed more than the target that has 32.

Figure 6.12 shows the ratio of missed attestation flags per cluster of validators (non-DVT vs DVT clusters), where we can appreciate that there is a clear miss pattern across the three flags: the head flag gets missed 4.66 more times than the source flag, which then is missed 4.54 times more than the target flag. The bottom part of the figure also shows that, although not that different from each other, the DVT validators failed 0.7% more of head flags, 0.6% of source flags, and 0.4% of target flags. These differences generally come from the largest clusters of seven and ten DVT nodes, which, as expected, could add a larger delay to the first consensus and then the signature aggregation of the duties.

On the other hand, figure 6.13 shows the same missed attestations split by flags but for the later Charon version v0.15.0. The figure shows a similar pattern of missed flags as with v0.14.0. However, we do appreciate a slightly lower difference between the non-DVT validators and the DVT clusters. In this newer version, the differences get reduced to 0.5% of missed head flags, 0.6% missed source flags and 0.4% missed targets, which means no major or very few upgrades were applied to the cluster attestation duties during the upgrade.

Inclusion Delays As previously mentioned, the inclusion delay clearly has a significant presence on the validator's performance, as it can draw the line on whether the flag is correct or not, regardless of the value itself. In that regard, figure 6.14 shows the cumulative distribution function (CDF) of the inclusion delay of the attestations produced by the DVT validators (aggregated by version, as there was no sign of significant improvements over the version upgrade). The figure showcases that over 72% of the attestations had an inclusion delay of one slot, which is the ideal scenario. However, the figure also shows that there is a 20% of attestations that stay with an inclusion delay between two and five slots, and the resting 8% beyond the five slots mark. We can observe that the three clusters show the same behaviour and percentiles. Although the matching of the CDFs across DVT clusters is not appreciated in the figure, the three CDFs are displayed in the figure, showing an overall similar performance.

Block Proposals At every slot of the chain, a single random validator is chosen to propose a new block, which includes all the "non-tracked-yet" attestations and sync committees broadcasted over the previous slots. As shown in figure 2.5, the block proposal is the first event that should occur inside a new slot. The protocol

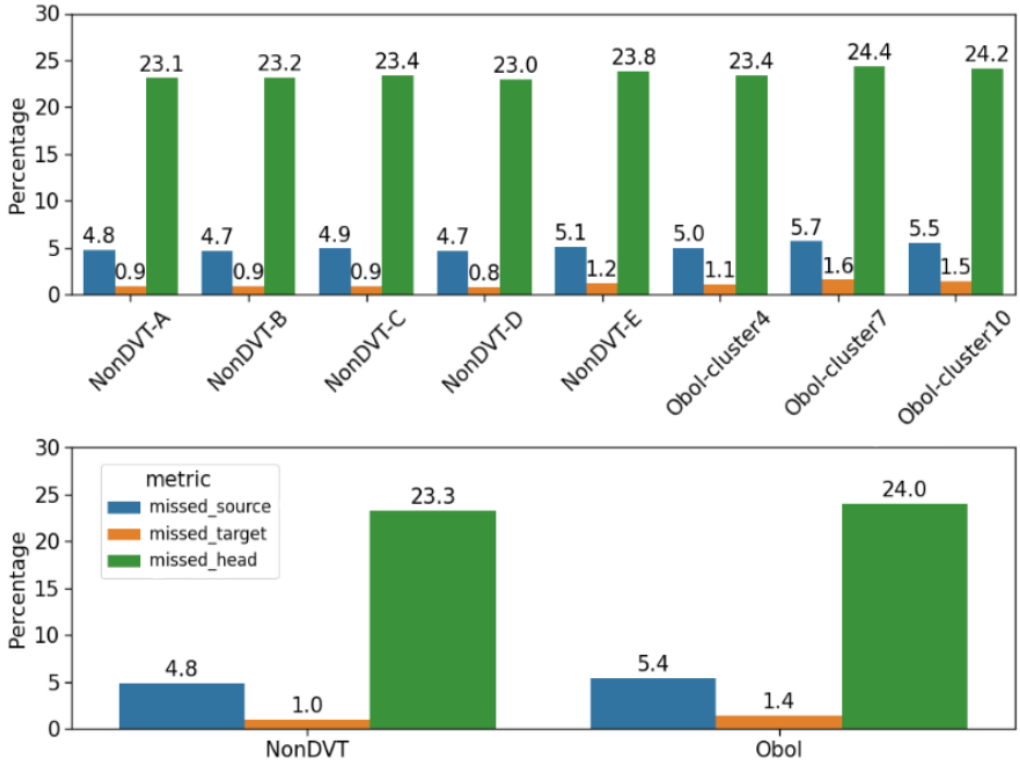

FIGURE 6.12: Percentage of missed attestation flags for the first part of the experiment with Charon on its version $v0.14.0$. The upper figure shows the missed flags across the different validator sets. In the bottom one, the missed flags are aggregated by types of clusters (non-DVT vs DVT/Obol).

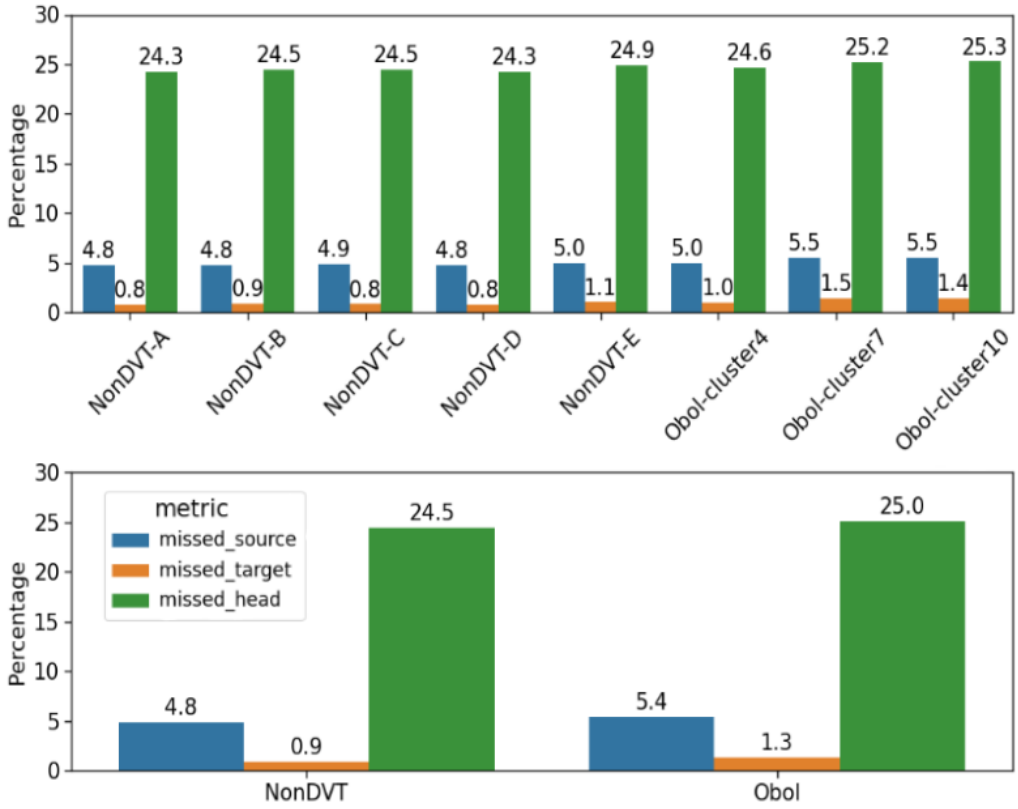

FIGURE 6.13: Percentage of missed attestation flags for the first part of the experiment with Charon on its version $v0.15.0$. The upper figure shows the missed flags across the different validator sets. In the bottom one, the missed flags are aggregated by types of clusters (non-DVT vs DVT/Obol).

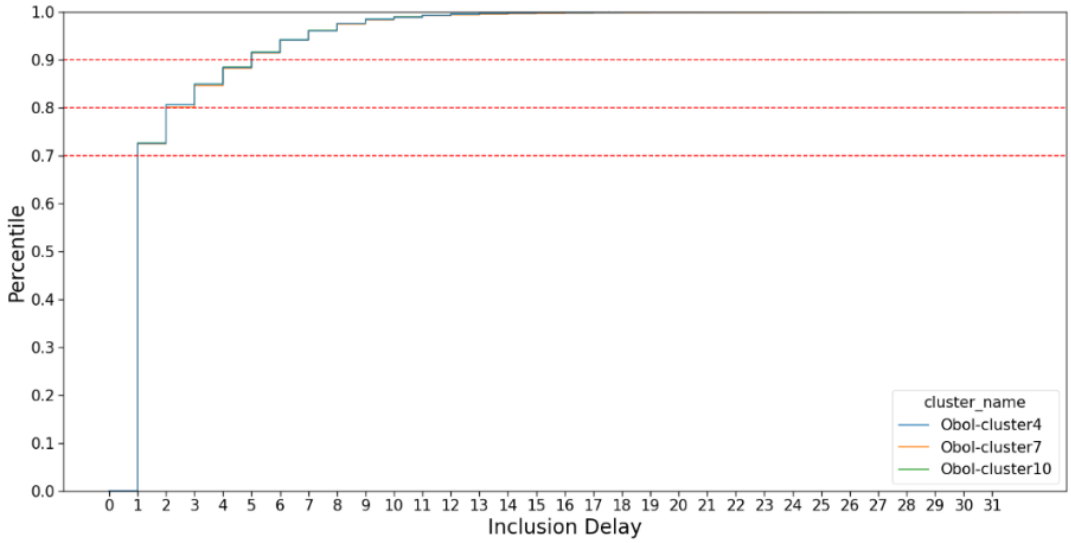

FIGURE 6.14: Cumulative distribution function of the attestation inclusion delay per DVT cluster, with both versions aggregated.

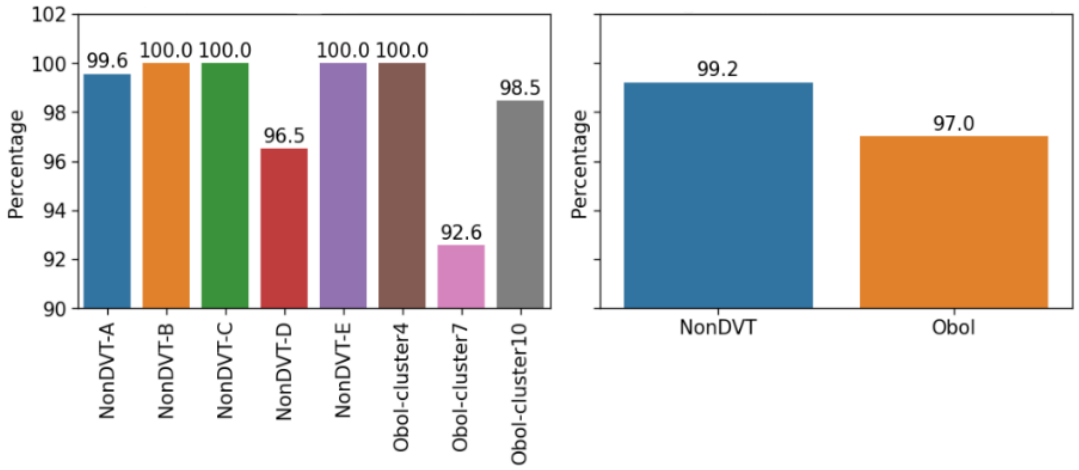

FIGURE 6.15: Percentage of successful block proposals of Charon on its version $v0.14.0$. On the left, the successful block proposals are aggregated by cluster. On the right, they are aggregated by validating technology.

leaves a security window of 4 seconds to propagate the block over the network; thus, small variations on when the block was first broadcasted might compromise the actual block proposal. This means that a block is only accepted when the 1/3th of the network receives the block proposal within the expected 4 seconds. In the case of the DVT validator clusters, there is a risk where the signature aggregation could extend the initial block propagation to the rest of the validators, exposing the cluster to a missing block. It is, therefore, very important to ensure that DVT does not add any critical delay when submitting a new block if they are chosen as block proposers.

In figure 6.15, we can observe that all validator sets had a successful block proposal rate above 92%. In fact, six of the validator sets stayed at the 98.5% or over it, except for the non-DVT client D and the Obol cluster7 that stayed at 96.5% and 92.6%, respectively. On the right part of the figure, the aggregation of the successful block proposals per type of validators show that the non-distributed validators still managed to get an extra 2% of proposed blocks. It is worth mentioning that the missing blocks are concentrated in the biggest clusters, cluster7 and cluster10, while the validators in cluster4 did not miss any block proposal. In particular, cluster7 struggled because two out of the seven nodes of the cluster had poorer connections to the rest of the cluster participants. However, this is something that we will further discuss in section 6.2.5. Either way, in both sets of validators, non-DVT and the DVT, non-DVT client D and cluster7 behaved like outliers.

In contrast to the missed attestation performance, the upgrade of Charon to its version $v0.15.0$ did impact the number of successful block proposals. Figure 6.16 shows the same data as in figure 6.15 but after upgrading the Charon client. Our measurements perceive an improvement of 2.7% in the number of proposed/missed blocks in distributed validators. While the non-DVT client D got stuck at 96.4% of block proposals, cluster7 achieved a solid 99.1% of the proposals, showing a big improvement in what we think is related to the internal cluster connectivity.

Due to the randomness of the block proposer election, a two-month study might not provide as many data points as we would desire, making the dataset susceptible to small variations or outlier scenarios. Table 6.8 shows the entirety of the missed blocks by cluster of vlaidators. In the table, as well as in the right side of figure 6.16,

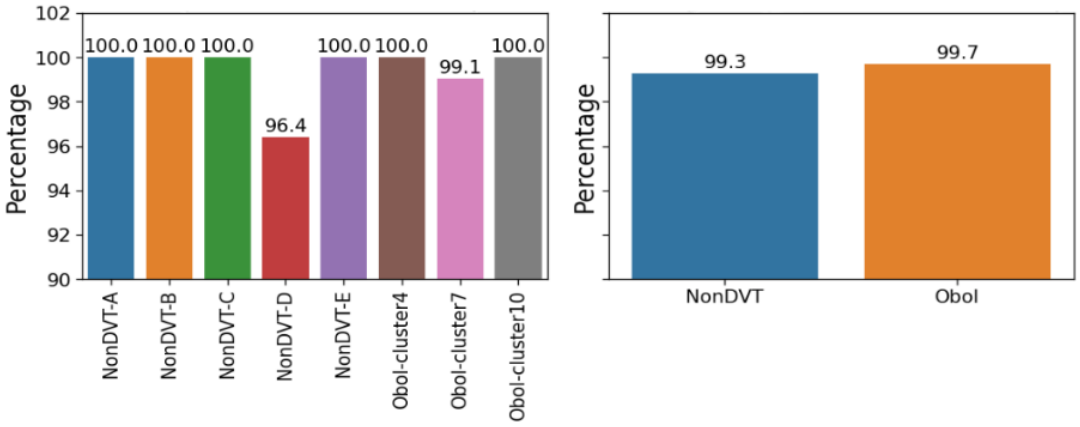

FIGURE 6.16: Percentage of successful block proposals of Charon on its version $v0.15.0$. On the left, the successful block proposals are aggregated by cluster. On the right, they are aggregated by validating technology.

		v0.14.0 (5500 epochs)		v0.15.0 (4500 epochs)	
Validator Set	**Validators**	**Proposed**	**Missed**	**Proposed**	**Missed**
NonDVT-A	600	225	1	176	0
NonDVT-B	600	235	0	208	0
NonDVT-c	600	240	0	191	0
NonDVT-D	600	222	8	188	7
NonDVT-E	600	216	0	203	0
obol-cluster4	1000	363	0	341	0
obol-cluster7	1000	361	29	313	3
obol-cluster10	1000	382	6	294	0

TABLE 6.8: Table summarizing all the exact block proposal duties per validator cluster, including the number of missed blocks over the whole study.

we could observe that the DVT validators got a higher ratio than its canonical alternative non-distributed validators over these last 4.500 epochs of the experiment. Despite the difference, which might sound exciting, it is clear that the difference comes from one particular that performed poorly during that period, bringing the mean performance down for NonDVT in general. However, this showcases that block proposals indeed do not get affected by the extra latency on version $v0.15.0$.

MER Ratio Achievement The performance of a validator in Ethereum's PoS heavily depends on the correctness of its duties. In fact, the 61% of the rewards a validator produces are directly associated with the attestation rewards, while the resting 29% are related to more random variables such as block proposals, sync committees, and fees from the included transactions at the execution level.

To quantify the profitability impact of the previously presented points, we measured the reward of each validator and compared them with their respective MER (check section 5.3 for further details). Figures 6.17 and 6.18 show that for all the validator clusters on the Charon's versions $v0.14.0$ and $v0.15.0$, respectively. The first one already shows that the profitability doesn't differ much from non-DVT validators to DVT ones. On Charon's $v0.14.0$ version, the difference only reached 0.6% in

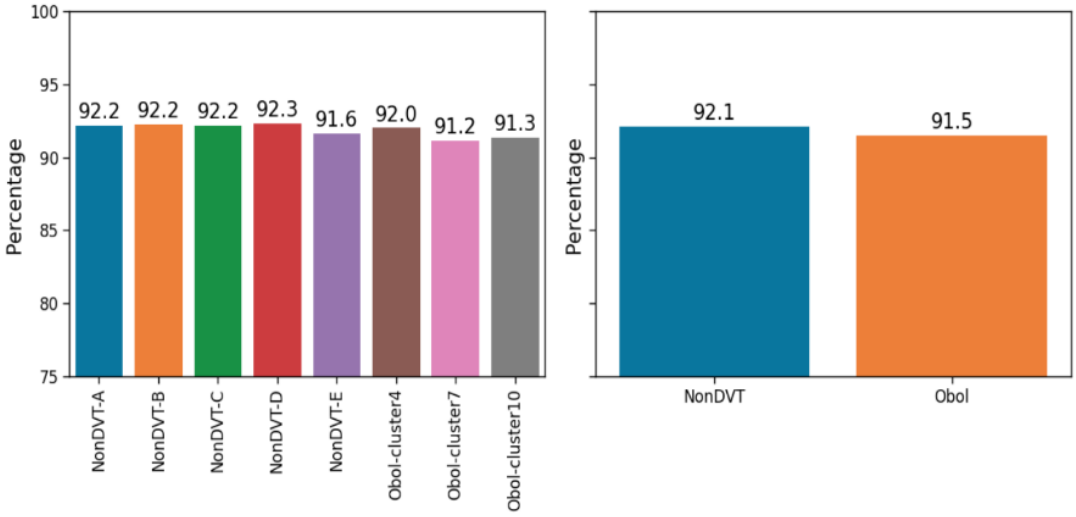

FIGURE 6.17: Ratio of achieved rewards with the MER of the validators aggregated by cluster on the left and by validator technology on the right on Charon's $v0.14.0$ version.

favour of non-DVT validators. On the other hand, the differences got narrower with the version upgrade, reducing the differences to a negligible 0.4% considering the extra resilience that DVT provides.

DVT Isolated Performance

The previous section 6.2.5 compared the performance of DVT validators versus its natural canonical competition. However, many relevant questions remain for the Ethereum staking community, i.e., in a staking ecosystem where over the 66% of the validators belong to staking entities or staking pools[10], how many validators could these large entities run on a single cluster?

To answer this and another similar question about the limits of DVT clusters, we set up a second study to spot any existing limitations.

Cluster Internal Latency Clustarization We have previously introduced the extra delay each DVT cluster adds to each duty's broadcasting. As expected, the performance of the DVT cluster heavily relies on the connectivity of the participating nodes. The lower the latency across nodes, the lower the delay, and the higher the number of participants, the higher the delay, as more signature aggregations need to be done.

To measure the internal connectivity between the spawned DVT nodes, figure 6.19 shows a heat map of the latency that each peer got with the rest aggregated by location and region. In this case, the figure shows the connectivity between Charon nodes using $v0.14.0$, where we can highly appreciate the clustering of low latency that peers get with others in the same region.

After upgrading to version $v0.15.0$, figure 6.20 showcases that the latency in the worst-case scenarios (very far distance between peers) has improved substantially. For example, the latency between Strasbourg and Singapore has decreased from 470ms in $v0.14.0$ to 300ms in $v0.15.0$, representing a 37% improvement. Similarly,

[10]https://monitoreth.io/nodes#active-validators-entities

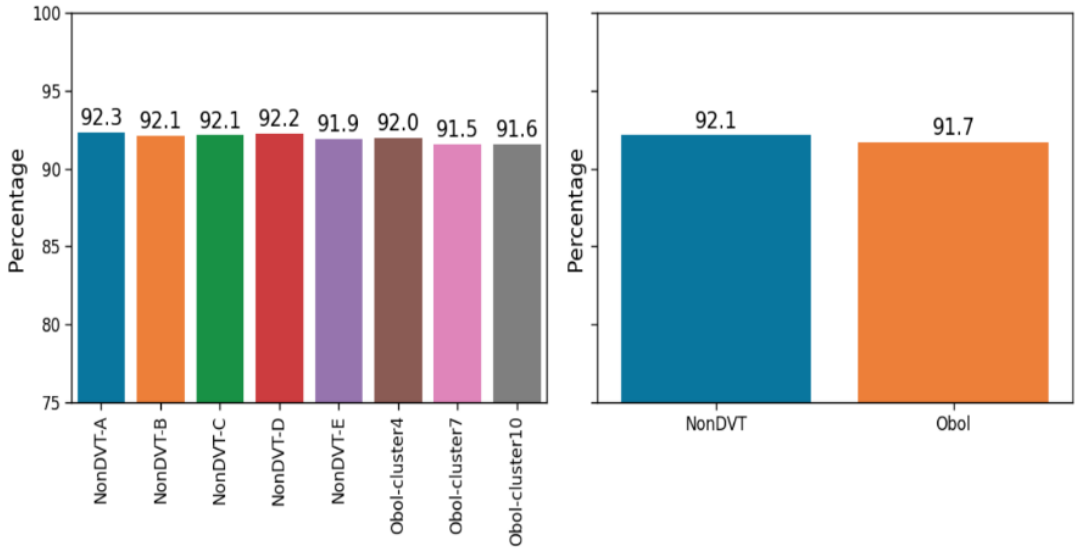

FIGURE 6.18: Ratio of achieved rewards with the MER of the validators aggregated by cluster on the left and by validator technology on the right on Charon's v0.15.0 version.

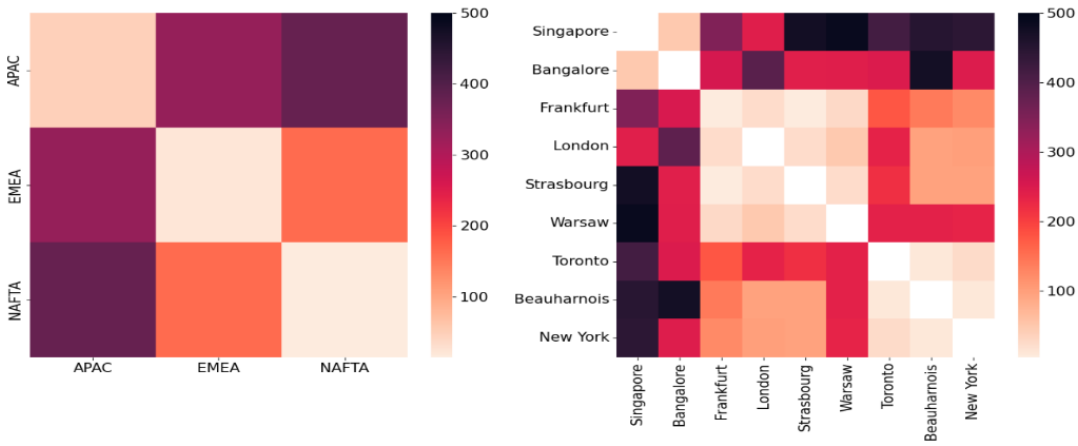

FIGURE 6.19: Latency heatmap across the different cluster nodes using Charon on its version 0.14.0. On the left, aggregated by regions: APAC (Asia–Pacific), EMEA (Europe, the Middle East and Africa) and NAFTA (North American Free Trade Agreement). On the right, aggregated by city.

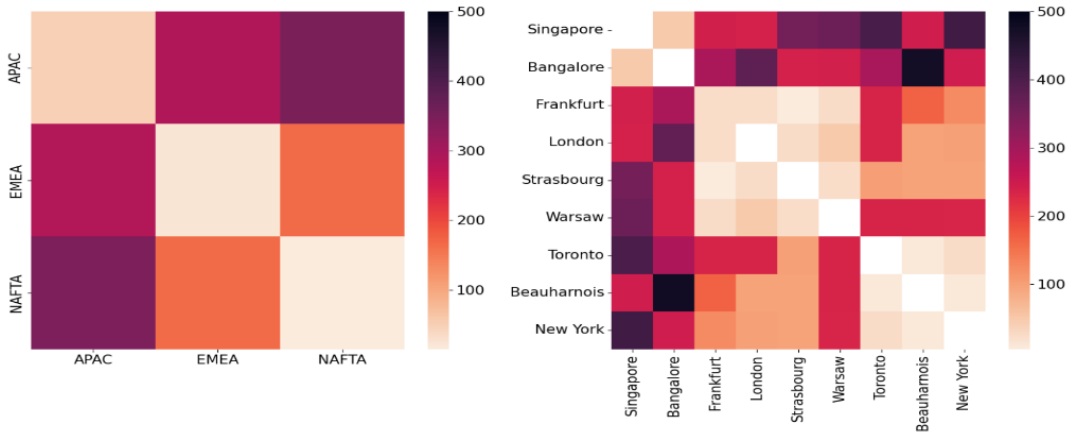

FIGURE 6.20: Latency heatmap across the different cluster nodes using Charon on its version 0.15.0. On the left, aggregated by regions: APAC (Asia–Pacific), EMEA (Europe, the Middle East and Africa) and NAFTA (North American Free Trade Agreement). On the right, aggregated by city.

the latency between Beauharnois and Singapore has decreased from 448 in v0.14.0 to 245ms in v0.15.0, which represents a 46% improvement.

These figures explain why the DVT cluster7 got more missed attestations in section 6.2.5; adding two nodes on a different region to the rest of the cluster significantly improves the connectivity.

6.2.6 Nodes' Performance

In addition to the previous setup, we performed a separate experiment to measure Charon's validator hosting capabilities. This experiment consisted of scaling the number of validators that are run on a cluster, with the main goal of analyzing how the Charon client handles such a high load of duties to perform. This study combined the 3.000 validators into a single cluster (cluster4 setup). However, we performed it in three different rounds:

- four-node cluster with 1000 keys for 1000 epochs
- four-node cluster with 2000 keys for 1000 epochs.
- four-node cluster with 3000 keys for 1000 epochs.

The same machines were reused for all steps. However, we had to clean them up when changing the number of validators because adding key shares to an existing cluster is impossible. Instead, it is necessary to run the DKG process to generate all the key shares used in the cluster.

Limit of DVT Validators per Cluster

A larger number of validators per cluster means more overhead to the hosting machine, as more duties must be performed. Remember that for every validator duty, the beacon node has to create the corresponding attention, which then needs to be signed by the validator client, which goes back to the beacon node to broadcast it. DVT adds extra steps for each of the duties as cluster participants first need to reach

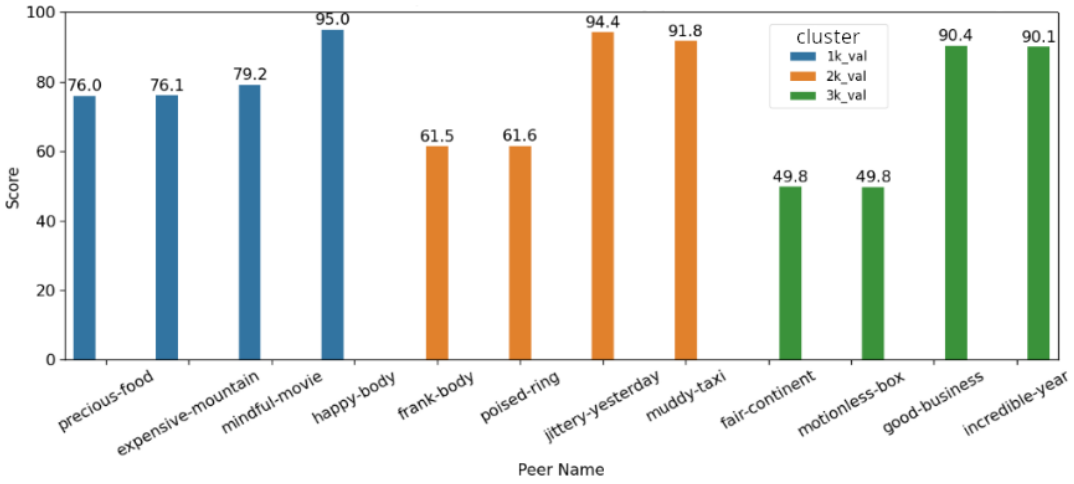

FIGURE 6.21: Beacon score of each Charon client participating in the four-node cluster per step in the experiment.

consensus through QBFT and then aggregate the signatures before sending the complete duty.

DVT cluster's beacon score As the first performance indicator of the overhead of increasing the number of validators per cluster, figure 6.21 presents the beacon node score of the peers that form the four-node cluster during the three phases of the experiment (each with a different number of validators). The figure shows that the more validators hosted in the cluster, the lower the beacon node score gets. On average, Obol's DVT cluster obtained a beacon node score of 81.5% with 1.000 validators, 77.3% with 2.000 validators (around 4% worse), and 69.9% with 3.000 validators (more than 11% worse compared to 1.000 validators).

Obtained rewards While the beacon node score gives us some indication about the performance of the Obol DVT cluster, there is no better benchmarking than measuring the rewards obtained during the three phases. As the number of validators increased with every step of the experiment, table 6.9 summarized the achieved rewards by the whole cluster, normalized by the number of validators performing duties during that period. The table shows an increase in the normalized achieved reward per validator as the number of validators increases.

Despite the unexpectedly decreasing beacon node score, rewards increase as the number of validators increases. On the one hand, this shows that Charon's internal beacon node score has a wide window for improvement. On the other hand, by analyzing the attestation flags (see table 6.10), we noticed a significantly lower number of source and target attestation flags missed when the number of validators increased.

This phenomenon is still not fully understood, and more experiments are necessary to reproduce and investigate these results. However, one fairly assumable cause of this unexpected behaviour is the impact of Goerli-related instabilities rather than a better cluster performance as the number of validators increased or a miss-performance of the client right after the new validator partial keys were added to the cluster.

Vals.	Total (ETH)	Per Validator
1000	9.424435	0.009424
2000	19.403201	0.009702
3000	30.437587	0.010146

TABLE 6.9: Rewards during scaling experiment with and without filtering block proposals and sync committee rewards.

	Total			Per Validator		
Vals.	Missed Source	Missed Target	Missed Head	Missed Source	Missed Target	Missed Head
1000	52320	12904	236131	52.320	12.904	236.131
2000	80770	18546	446948	40.385	9.273	223.474
3000	103889	22413	602982	34.630	7.471	200.994

TABLE 6.10: Number of missed attestation per different validator set.

6.2.7 Conclusion

This section presents an exhaustive empirical study of the Distributed Validator Technology. The provided analysis has shown that there aren't significant differences in validators' steadiness and profitability while comparing a DVT validator cluster to a canonical validator setup.

The presented measurements have shown that the difference in missed attestations between running a DVT cluster and using a standard setup is lower than the 1% in favour of the standard validator software. However, in any case, the DVT validators achieved a steady ratio over the MER of over 91%, which results in a mean difference of 0.4% lower than its counterpart. As the experiment was performed on a testnet, which generally represents a harder network environment for validators, the presented results represent one of the hardest witness-able scenarios of a network like the mainnet beacon chain (real economic incentives make the network more steady).

We want to remark that, even in a scenario where the profitability was slightly lower, the extra reliability that DVT validator clusters could bring to staking entities or node operators could be negligible if we compare it with existing slashing risks that canonical staking can have. The technology has shown enough maturity to support up to 3.000 validators per cluster, which enables the possibility of running DVT clusters at the node-operator level. Furthermore, we must introduce an extra advantage point for the DVT, which helps lower the entry barrier for staking in Ethereum, as users could participate without having 32ETH.

However, it is worth remarking that, based on this study's profitability and latency results, we highly recommend keeping the DVT cluster in the same region. This would significantly improve the cluster's performance, achieving the full resilience that DVT aims to offer. The study has also provided a clear overview of the evolution of DVT. One single version upgrade has meant a significant performance improvement in the block proposals and internal cluster connectivity. This means that technology still has a lot of room for improvement.

To summarize, this section showcases the significant contribution of DVT to Ethereum's staking ecosystem, providing resilience to the validator deployment setups and achieving a remarkable performance compared to existing, more mature canonical validator software.

6.3 Conclusion

Although the effects that the location of a node could have on its latency with the core of the network was expected, this chapter described and quantified the impact of this one with the performance of a validator, identifying the implication of all the involved parameters:

- The apparition of third-party protocols, such as DVT or MEV relays, adds more delay to block broadcasting and arrival times.
- Nodes located further from the network's core experience higher downtime ratios as they are subjected to more chain reorganizations due to the higher network latency.

However, the chapter presents some tweaks and recommendations that users or node operators could apply to reduce the disadvantageous situation that having extra latency means:

- There is a high link between the hardware resources and the experience node's downtime. The higher the latency, the more users would have to overestimate the hardware. The likelihood of receiving later events like blocks or attestations has to be compensated with a faster, more capable CPU, disk, and memory.
- In the case of DVT, respecting the extra hardware that runs an additional client means that as soon as the clients in the cluster are at least regionally closer, the cluster shouldn't experience any disruption. In this case, due to the community/cluster's implication on making the cluster pairing, we highly suggest locating nodes on locations that respect the 250 milliseconds margin between each node.
- Those new technologies of external protocols, despite adding some range of delay, are still in a premature phase. We've identified significant improvements just on a single DVT software version upgrade. Thus, as these protocols mature, their implication on the chain's finalization is expected to be lower.

Chapter 7

Exploring the Future of Ethereum's Scalability

7.1 Introduction

Scalability has proven to be one of the major challenges in blockchain technology, at least when trying to keep it decentralized and secure [212], and Ethereum [62] is not different from the rest in that respect [33]. To tackle those scalability limitations, Ethereum plans to rely on the so-called Layer 2 (L2) solutions [139] [175], where we can find solutions such as private payment channels [142] or the recently popular roll-ups [189]. These latest protocols, which include mature projects such as Polygon[1], Arbitrum[92], Optimism[2], or zkSync[3], benefit from the speed of processing transactions off-chain. L2 solutions optimize their operation costs by aggregating multiple transactions into smaller ones, providing, in some cases, a set of proofs or commitments that can be used to verify the correctness of the operations. However, all of them still rely on the consensus and security of layer 1s to ensure that the interactions are traceable and immutable.

Among the available solutions, roll-ups have been popularised over the last years for their similar yet cheaper EVM-compatible solutions. However, these protocols generally need L1's chain space to store the commitments of the processed data. Ethereum researchers have proposed data-sharding, colloquially known as "DankSharding" [45], to facilitate that needed space for roll-ups. The idea is to offer larger, cheaper, and off-chain block space (blobs) to fill up with verifiable commitments data[4]. The nature of roll-ups generally relies on fault or verifiable proofs as a mechanism to validate transactions. This means that every time roll-up submits a bunch of transactions to the main chain, also sharing the respective verification data as a blob, the rest of the network will have X number of weeks to fetch and verify if something went wrong. Under this premise, transactions could be verified as one honest verifier exists on the network during those weeks. But those proofs are not needed in the long term [56]. Therefore, blob data can be considered ephemeral and discarded after a certain time [155]. This protocol proposal will scale the execution payload size, as after the execution payload of a block, the related "blobs" will also be shared on separate GossipSub [199] message broadcasting topics. This makes the total $block + blobspace$ increase from the current average of $150KB$[5] to something in the order of $32MB$.

[1] https://polygon.technology/papers/pol-whitepaper
[2] https://www.optimism.io/
[3] https://zksync.io/
[4] https://domothy.com/blobspace/
[5] https://etherscan.io/chart/blocksize, https://ethresear.ch/t/big-block-diffusion-and-organic-big-blocks-on-ethereum/17346

The proposal relies on Erasure Coding [206] and Data Availability Sampling (DAS) [75] techniques to split the blob block into segments/chunks/samples to reduce the bandwidth, storage and computational overhead that process all blobs imply for nodes and validators. Splitting and sharing the blob block in smaller pieces makes the propagation of blobs easier through the network, as the validators will only have to subscribe, receive, and process a smaller portion of the larger coming blob block. Ordinary (non-validator) nodes instead rely entirely on probabilistic sampling, receiving only a few segments of the block to verify that the block is reconstructable using the erasure code with very high probability. The challenge of increasing the block size from an average of $150KB$ to $32MB$ is not simple. The upgrade to allocate more blobs in a cheaper ephemeral block space has many configurations and parameters that must be adjusted to make it "feasible". Figure 7.1 represents the proposed Execution Layer block's division for DAS, where the block will be first split into 256 rows and columns and then equally extended in both directions (rows and columns) applying Reed-Solomon Encoding [206] to ensure each row and column (and therefore the entire block) can be reconstructed as long as half of extended row or column samples are retrievable.

Due to the synergies between the existing blob samples' addressing needs from Ethereum's DAS and the content addressing capabilities of DHTs, this chapter presents the results of simulating a Kademlia-based DHT for Ethereum's data-sharding case. The chapter explores the time restrictions of common DHT operations under the expected DankSharding workload. We present a configurable DHT simulator that can reproduce the DHT operations under connection errors and network latencies, which is essential to accurately recreate not that foreseeable peer-to-peer networks. Furthermore, We raise concerns about the scalability limitations of DHTs like IPFS's when they suffer a high throughput of store and retrieval operations. Especially when the protocol wants to keep the hardware resources close to commodity home hardware as Ethereum does to promote decentralisation by allowing home node operators. Considering the outcome of this research, we propose a set of alternatives that could potentially overcome the listed limitations, ensuring low overhead for the nodes while still providing the expected requirements for blockchain scalability.

7.2 How does a DHT fit in Ethereum's DAS approach?

The challenge of increasing the block size from an average of $150KB$ to $32MB$ is not simple. The upgrade to allocate more blobs in a cheaper ephemeral block space has many configurations and parameters that must be adjusted to make it "feasible". Figure 7.1 represents the proposed Execution Layer block's division for DAS, where the block will be first split into 256 rows and columns and then equally extended in both directions (rows and columns) applying Reed-Solomon Encoding [206] to ensure each row and column (and therefore the entire block) can be reconstructed as long as half of extended row or column samples are retrievable.

With the current count of over 950.000 active validators[6] distributed around 13.000 active Beacon Nodes[7] (at the moment of writing this chapter), DAS aims to disseminate the block by entire rows and columns through the usage of distinct GossipSub topics [199]. Chapter 5 presented how each active validator will have to attest to a set of two random rows and columns in a single slot in an epoch. This will ensure optimal bandwidth-efficient block propagation. However, since part of the network's

[6] https://ethseer.io/?network=mainnet
[7] https://monitoreth.io/nodes#act-nodes

privacy relies on not knowing which node in the network hosts which validator (preserving the anonymity of the validators), it complicates checking which sample of the blob block is available on which node without compromising the privacy of the validators, as the topic subscription of each node could be correlated with the identity of validators.

In this privacy-preserving case, the proposed DAS schema still needs a resilient and privacy-preserving method able to route any blob verifier to the node that keeps it in the network, and this is where a popular DHT such as Kademlia enter into place. The Kademlia DHT has been widely used for content addressing by many networks such as IPFS [17], where it has proven to ensure resilience against the constantly changing network while still providing fast content retrieval times and some privacy-enhancing solutions like “double hashing" [188] of the stored records. With all these key features, a Kademlia DHT is a suitable candidate to provide available samples to any interested node, as the seeding of the DHT considerably reduces the possibility of unveiling the location of validators. In this case, instead of seeding the DHT with the “Provider Records" (pointers to the content provider in IPFS), because the size of each sample (as introduced in figure 7.1) is relatively small (512 bytes of data plus 48 bytes of KZG proofs [104]), we propose using a Kademlia-based DHT to evenly distribute the samples based on the Beacon Node’s ID and the segment’s distance. This would reduce the number of operations to retrieve each sample from two: i) ask the network who has the content, and ii) contact and download the sample from the node, directly to the second step. However, due to the highly demanding DAS approach of DankSharding, the limits of the Kademlia DHT still remain unknown.

7.3 Methodology

Ethereum’s DAS approach presents an ideal yet ambitious synergy with DHTs. These networks offer light and resilient logical links across participants that can resist the sometimes chaotic dynamics of peer-to-peer networks. However, its direct application in such a demanding protocol as DankSharding remains unknown. This section introduces the methodology we’ve used to simulate a DHT network with most parameters involved in a peer-to-peer network and under a heavy workload. Furthermore, we also introduce the developed framework that helped us validate the results of the simulations on the live IPFS network.

7.3.1 DHT Simulator

Since the internal parameters of the possible DHT are not defined (at the moment of writing this chapter), the chapter explores a possible DHT implementation for DAS through a simulator. The DAS simulator relies on a Python module that implements all the components of a Kademlia DHT called “py-DHT" [157]. The module allows the creation of a network of the given size, initializing the routing table of the nodes while using the XOR binary distance between nodeIDs to fill up the K-buckets. Inside the module, the two main components, the DHT network and the DHT nodes (also known as DHT clients), have different purposes.

The network module offers a common shared perspective for the simulation, as shown in figure 7.2. It keeps track of all nodes and simulates the communication

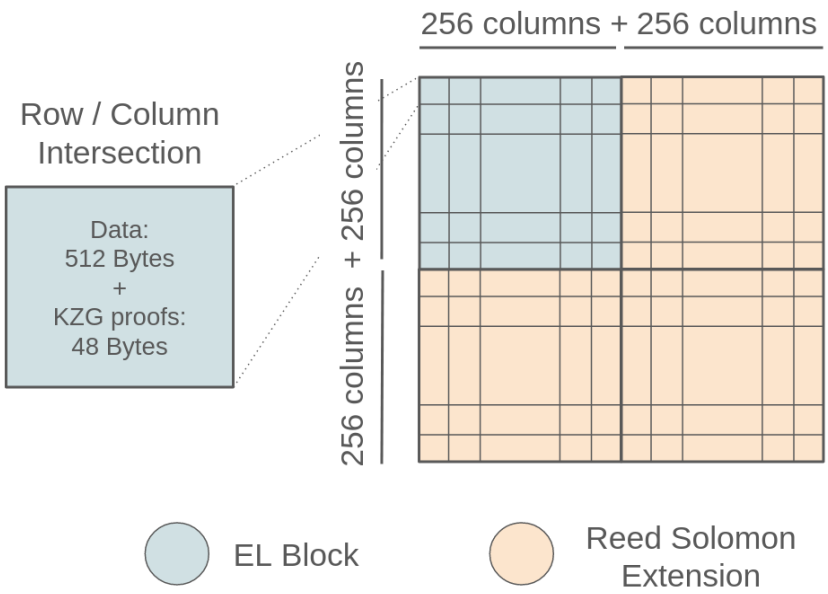

FIGURE 7.1: Block split proposal for DankSharding's DAS.

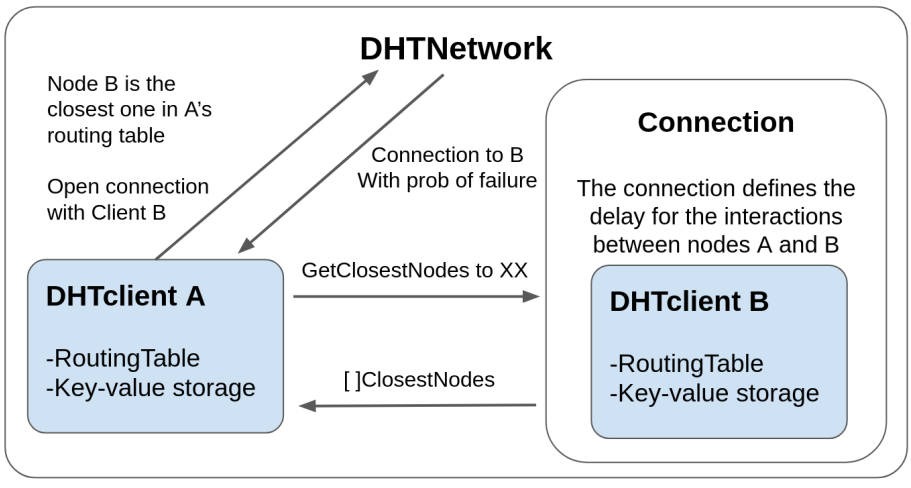

FIGURE 7.2: Description of the DHT components in the *dht* simulator, and an example of the internal logic of their interaction.

between two nodes by introducing latencies and eventual losses. Because peer-to-peer networks tend to be subject to node congestion, node churn, or even connection churn [79, 44], the network configuration allows to define the following set of parameters that helps reproduce any network behaviour:

- Number of nodes participating in the DHT network.
- Fast error rate: ratio of connections that remote peers will refuse.
- Slow error rate: ratio of the connections that timeout when connecting a remote peer.
- Connection delay range: latencies applied to the communication between two nodes.
- Fast delay range: latencies applied when a *fast error* occurs.
- Slow delay range: latencies applied when a *slow error* occurs.
- Gamma: the incremental delay applied to the communication between nodes when a peer is contacted by others, simulating the overhead of handling multiple connections.

On the other hand, the DHT clients are the ones that populate the simulated network. They include all the logic of the DHT, allowing themselves to swap information if the connection through the network interface succeeds. The simulator recreates the performance of a DHT, performing events such as initialization of the routing table, updating the routing table, looking up the closest peers to a key, providing a given value to the network, lookup for a specific key. The DHT clients have the configuration of the Kademlia DHT, including the following parameters:

- K: replication factor for the DHT and k-buckets' size.
- Alpha: maximum number of concurrent connections a DHT client can have while looking for a key/value or the closest clients to a key.
- Beta: number of close clients a remote peer provides when looking for a key.

We refer the reader to [123] for the exact definition of these parameters. The "py-DHT" module extracts all the timing and accuracy metrics from each individual operation, allowing the inspection of the network's performance under certain circumstances.

7.3.2 Verification of the DHT Simulations

To validate the simulated results of the *dht* module, the chapter includes a comparison with the IPFS live network's performance. The results were validated by comparing the performance of the most basic operations in a DHT, such as DHT provides and DHT lookups in IPFS, with simulations replicating the same DHT and network parameters: $K = 20$, $Alpha = 3$, and $Beta = 20$. To obtain those performance measurements from the IPFS network, the team developed a dedicated tool, "looking-up-ipfs" [117], which provides measurements of the internal operations of an IPFS's DHT-host interacting with the live network. The tool can concurrently generate and provide a given number of random Content Identifiers (CIDs) to the network[8], dumping the duration of the process and the contacted peers while finding

[8]NOTE: when advertising content to the IPFS network, nodes only replicate the pointer to who has the content (Provider's Records) to the k closest peers to the CID's hash values.

the *K* closest peers of each content-provide operation. Once the CIDs are published, the tool tests the retrievability of the same CIDs later, tracking the individual duration and the total contacted nodes during each independent operation. Furthermore, as it attempts the retrieval concurrently, it also allows us to measure the overhead of performing multiple low-demanding operations from a single DHT host.

7.3.3 Development of the experiments

Each experiment presented in the following evaluation section 7.4 was conducted on dedicated hardware machines. The first part, the simulation of the DHT for Ethereum's DAS needs [51], was run on a PC with a Ryzen 5900X CPU, 32GB of memory, and 1TB of SSD disk. On the other hand, the IPFS benchmarks using the "looking-up-ipfs" tools were run on an AWS cloud machine in Paris with 4 CPU cores, 8GB of memory, and 50GB of SSD disk on the 3rd of October, 2023.

7.4 Evaluation

7.4.1 DHT lookups

One of the key operations in a DHT is its ability to find which node in the network has the value for a specific key. This operation is generally known as *DHT lookup*, and it consists of a set of recursive operations that, using the XOR distance between the node IDs and the given key, discover closer and closer nodes until, eventually, a node that has the value is found (assuming some node has it). DAS needs fast data retrievability from the DHT to ensure that blobs can be verified "fast enough"[9]. For this reason, measuring DHT's performance is critical to check if it would be a viable option for DAS in Ethereum. A peculiarity of DAS verification is that a node is starting concurrent lookup queries for several randomly selected samples. The DAS verification is considered complete when every single selected sample is retrieved. This section describes the main findings observed when simulating such concurrent queries on a DHT. We use 80 queries from a node, as finding 80 retrievable samples provides high enough probabilistic guarantees on a 512-by-512 segment 2D encoded block [46]. We also run similar concurrent queries on the live IPFS network. This serves as a validation for the correctness of the simulator, where the number of hops and the aggregated delay for the operation match a live network if the parameters are correctly set.

Lookup Hops A hop is an individual step in the recursive lookup operation. It implies asking a remote node in the network for a number (*beta*) of the closest peers it has to the given key or the value of the key if it has it. This is how the DHT narrows down the lookup of a value. Figure 7.3 shows the CDF of the number of hops done by an IPFS DHT node when performing 100 sets of 80 concurrent DHT lookups. The figure shows how the 99% of lookups were done in under 18 hops, with a small tail in the last 1% that can reach the 100 hops on some rare occasions. The numbers measured with the DHT simulator report similar results, as figure 7.5 shows. With a fast-failure rate of 10% of the connections, the simulation of 100 sets of 200 concurrent lookups report a 99th percentile of 12 to 14 hops. We also notice that the concurrency overhead parameter does not impact the number of hops, as we see

[9] There is an ongoing debate in the Ethereum community about how fast this verification should be. It is commonly argued that one slot time (12 seconds) is fast enough.

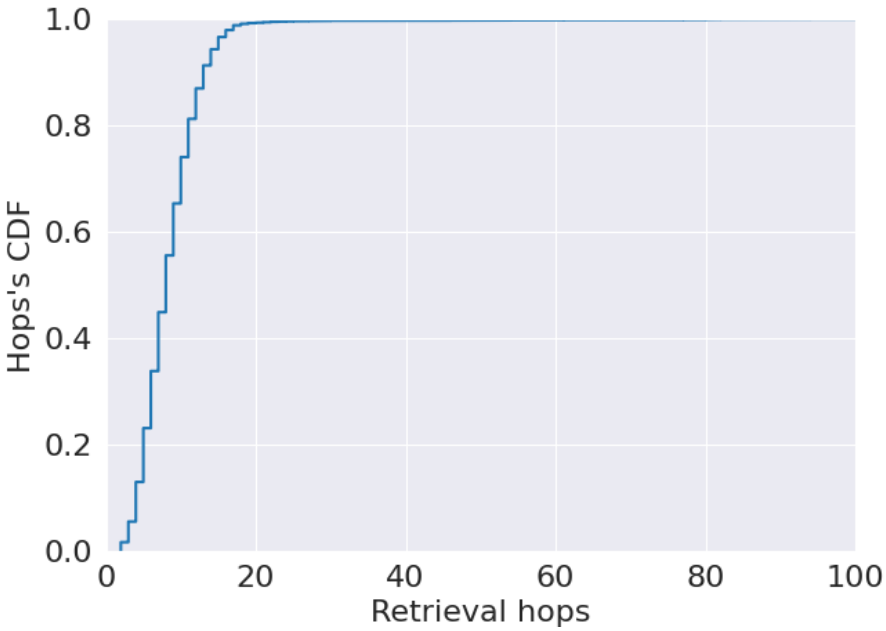

FIGURE 7.3: CDF of the DHT hops while performing 100 concurrent retrievals in the IPFS network.

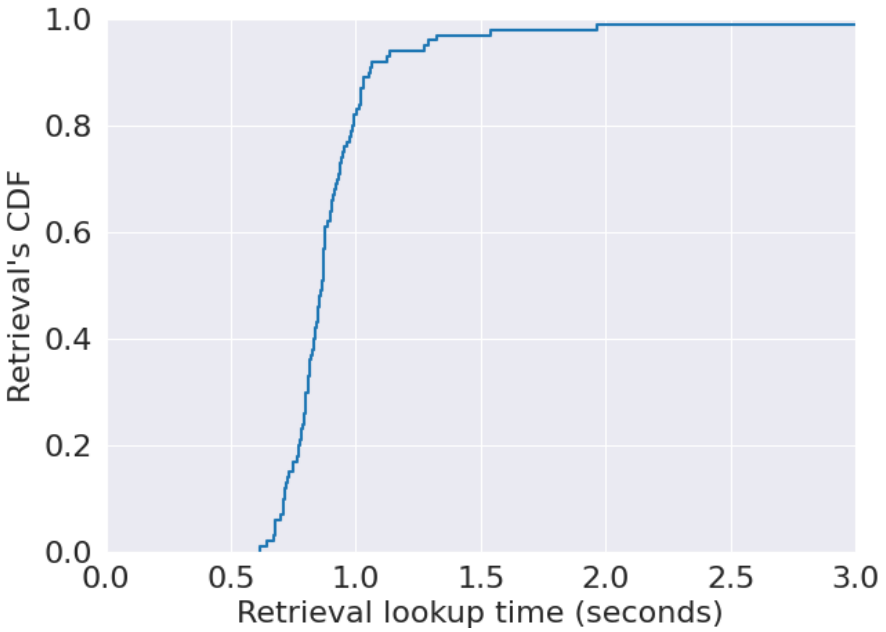

FIGURE 7.4: CDF of the time to finish 100 sets of 80 concurrent retrievals from the IPFS network.

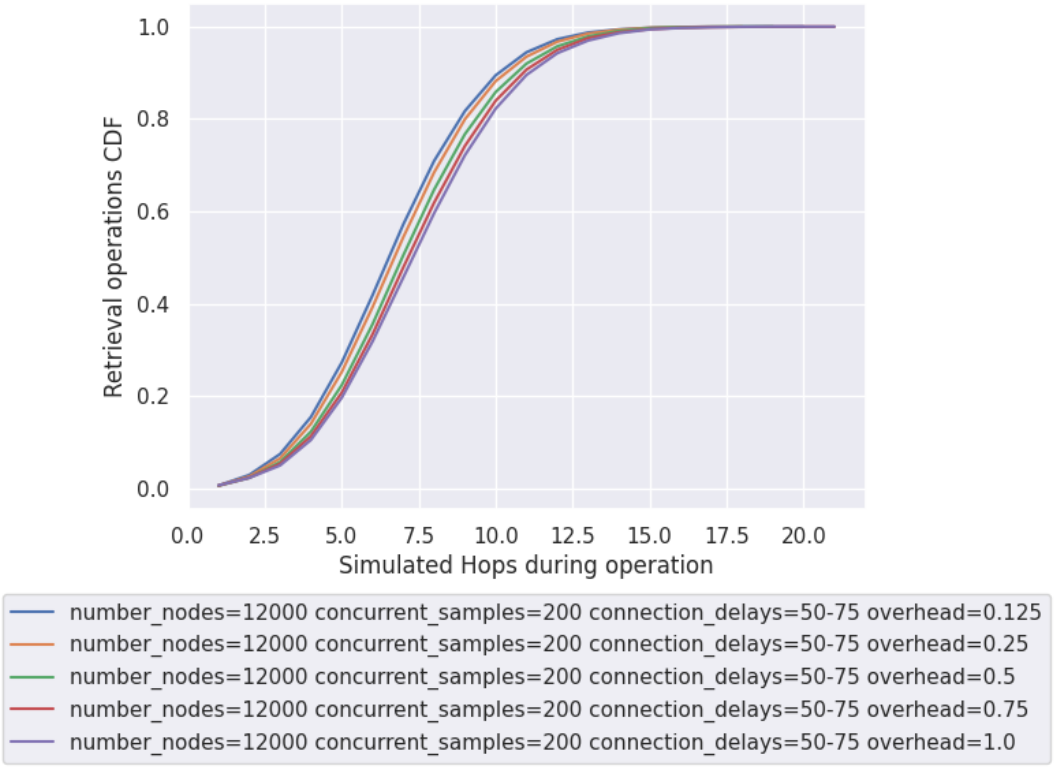

FIGURE 7.5: CDF of the simulated number of hops while looking for 200 concurrent keys with different overheads with $k = 20$, $alpha = 3$, and $beta = 20$.

a negligible difference between the distributions for different overhead values. This makes sense as the time it takes a node to answer does not change the content of its routing table, and, therefore, the number of hops is not impacted.

Time Performance Nodes join and leave the network as they please in permissionless networks such as IPFS and Ethereum. However, the users in the Ethereum network have shown more stability in maintaining their nodes for long periods. Users are generally staking or extracting on-chain data from nodes, so they are somehow incentivised to keep the nodes up and running. As a result, this stability can create healthier routing tables and potentially increase the DHT lookup time performance. In an attempt to validate the impact of the network conditions on the DHT lookup performance, figure 7.6 compiles the lookup time's CDF in the IPFS network done by the Probelab Team [152], with the respective estimated network conditions in each of the regions. It is not a surprise that different regions have different connectivity. The difference on the 90th percentile of the regions varies from 700 milliseconds to retrieve a value in the US and Europe to 1.7 seconds and 1.5 milliseconds in South America and Africa. Moreover, this connectivity difference can magnify in p2p networks where nodes cluster more in certain regions. The figure also showcases the possibility of configuring the *dht* module to fit any realistic network conditions. The figure shows that with the variation of the delay range at the simulated network, the results can accurately fit the measurements done in the IPFS DHT network from the lookup node's perspective.

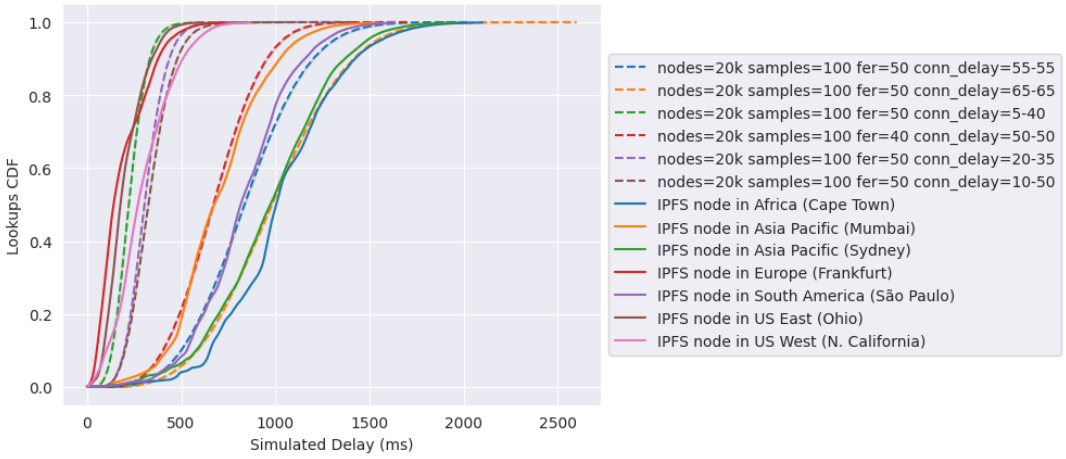

FIGURE 7.6: Accuracy of the DHT simulator model matching lookup performances in the IPFS network from different locations while keeping $k = 20$, $alpha = 3$, and $beta = 20$. Note that "fer" stands for "fast error rate".

Concurrency Overhead We've mentioned the differences between the IPFS and the Ethereum network. The nature of IPFS, where 80% of users altruistically spawn a node and disconnect it after 8 hours [193], makes it more subjected to node churn [44]. This generally impacts the performance, as these nodes might be present in the routing table of other nodes for several minutes. Despite the expected node churn in Ethereum is expected to be lower, having non-active nodes inside the nodes' routing table doesn't only affect the direct time performance of the DHT operations. The main solution to overcome a certain connection failure rate is to attempt multiple simultaneous ones, with the extra cost of handling those extra concurrent requests. The presented $alpha$ parameter helps overcome the time impact of facing unreachable nodes, which in IPFS' is set to 3 concurrent connections. However, it multiplies the cost of making a single lookup. Figure 7.7 shows the time impact of different concurrency overhead parameters on 200 concurrent DHT lookups from a single retrieving node's perspective. The overhead is applied to nodes contacted during the concurrent simulated operations. This means that each node connected during the lookup operation will apply an only increasing overhead delay to its connection delay as other concurrent operations contact the same peer. Suppose a set 200 connection needs to contact peer A on three occasions. In that case, the first won't have any overhead delay, the second one will have a penalty of the overhead parameter, and the third one will have a penalty of two times the parameter. Users with access to larger hardware resources could be less restricted by the concurrency overhead of retrieving multiple block segments in sub-second lookups. In other occasions where the hardware resources are more limited, like in a solo-staker scenario, the DAS operations will be more penalized for handling so many interactions. This exposes a clear disadvantage for smaller users if the number of lookups is too high in the long-term projection of the protocol.

To compare the existing overhead in the IFPS DHT client, figure 7.4 exposes the CDF time of concluding 80 concurrent lookups in the IPFS network in a set of 100 rounds. The figure shows that 90% of concurrent lookups performed between 600ms and 1.2 seconds. Compared to the numbers presented in figure 7.6, we can clearly see that IPFS's client does see an impact in performance originating from a concurrency overhead, not at least at this number of concurrent requests. Therefore, it is also

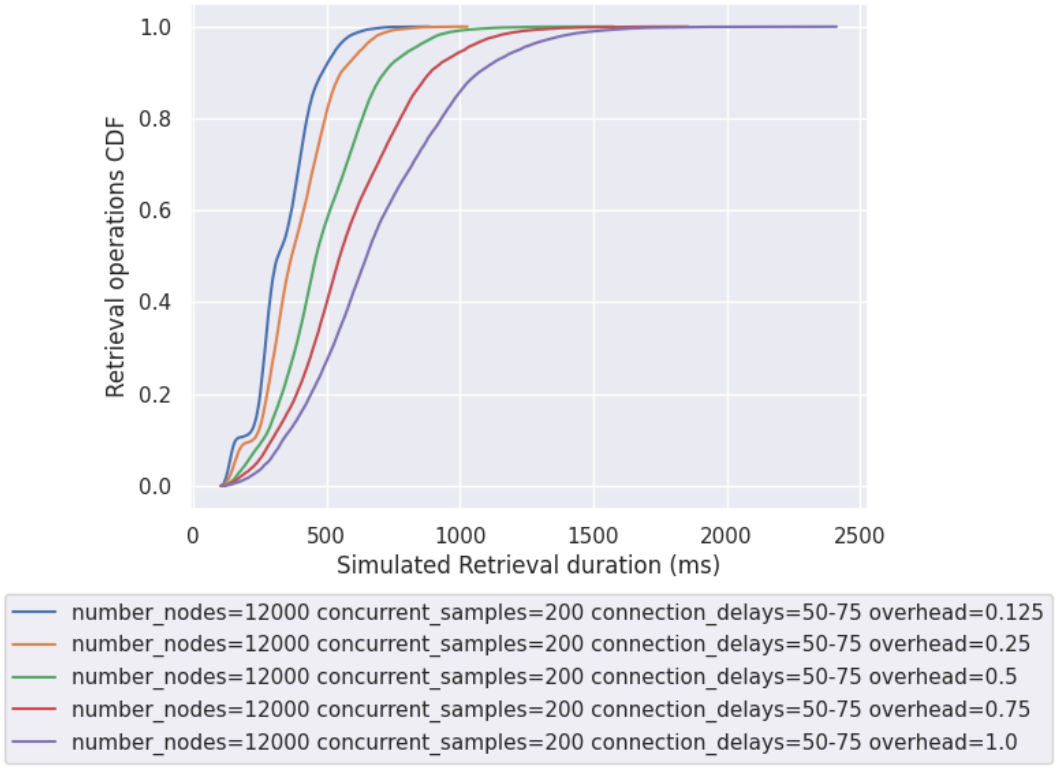

FIGURE 7.7: CDF of the simulated 100 DHT retrieval sets with 200 concurrent keys each using different overheads. The DHT simulation had the following parameters: $k = 20$, $alpha = 3$, and $beta = 20$.

expected to have an effect on a possible DHT for Ethereum's DAS.

7.4.2 Seeding of the DHT

The results of the experiments are clear: modern Kademlia-based DHTs can achieve sub-second sampling performance on its 90th percentile with the favourable conditions that the Ethereum network provides. Even if looking at less favourable conditions and at higher percentiles, we see latencies well below the slot time of 12 seconds. However, despite the logarithmic scaling solution for content addressing, the idea of the DHT seems to be reaching its limits in highly demanding conditions. While the sampling itself seems to work, the first step of a DHT solution is seeding the block data into the DHT, where the samples are dispersed around the network according to their position in the DHT address space. Ethereum's current network conditions report roughly 13.000 active nodes. Combined with the suggested standard Kademlia parameters [36] of $k = 20$ and $alpha = 3$, the seeding of 262.144 block segments (512 columns * 512 rows) would produce a ratio of 403.29 block segments that each node in the network would have to store, on average, every 12 seconds. Moreover, storing the block samples doesn't only affect the last contacted nodes. Providing the keys is a process that starts from a distinct lookup to find the closest k nodes to the segment's key, performing a median of 8 to 10 hops until the closest nodes are notified. If handled simply as a large number of individual storage requests, the overall network load can easily overload the network, considering that all this should happen quickly before the actual sampling begins.

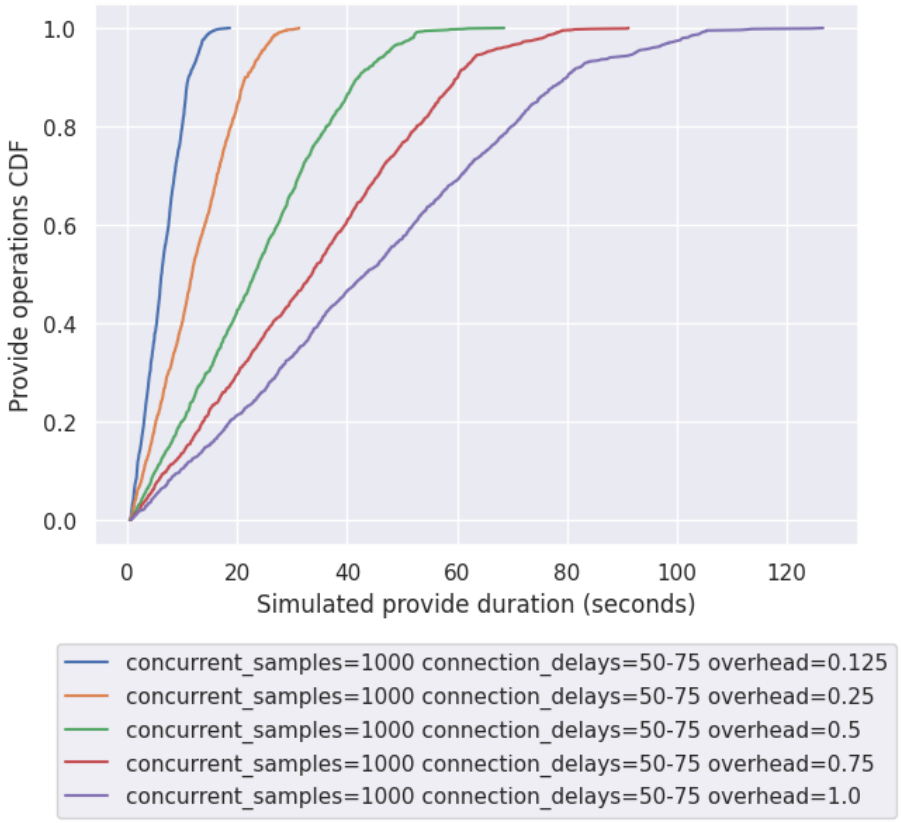

FIGURE 7.8: CDF of the simulated 1.000 concurrent DHT provide operations from a single node.

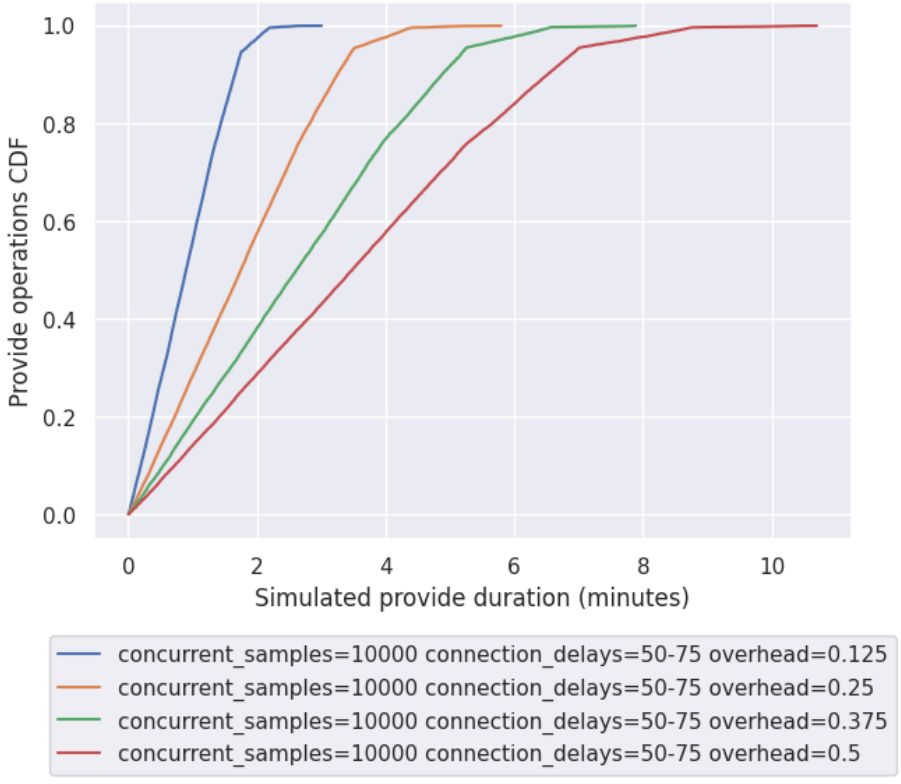

FIGURE 7.9: CDF of the simulated 10.000 concurrent DHT provide operations from a single node.

To simulate how tedious this seeding from a single node step would look in a Kademlia DHT, figure 7.8 and 7.9 show the CDF of the simulated DHT provide times for 1.000 and 10.000 samples concurrently in a network of 12.000 nodes. They show that the overhead factor, together with the value of $k = 20$, has a huge impact on the total duration of the operation. Figure 7.8 shows that a low number of samples, 1.000, can still be broadcast to the network within 20 seconds. However, it can only be achieved if the overhead generated in the network remains low, and existing users would need to upgrade their hardware to ensure enough resources to keep the DHT. As we increase the number of concurrently distributed samples from a single node (check figure 7.9 with 10.000 concurrent samples), the CDF shows that the DHT starts to find a bottleneck.

The biggest bottleneck comes from the routing nodes on the seeder's routing table. As Kademlia DHT keeps the most optimal and active nodes in the routing table using the XOR distances, the routing table barely changes in such a short time. Thus, those 250 to 300 in the routing table are always contacted first on each provided operation, being the most overwhelmed nodes in the network. This gets more visible if we simulate the provided operation from a single node (i.e., the block builder[154]) of the 262.144 samples inside a blob block. Figure 7.10 shows that even in a simulation of the best possible conditions (setting a very low factor of 0.015ms per connection), the congestion in the network would make the process delayed to the 10 to 14 minutes, existing 50 times the original broadcasting deadline of the 12 seconds. To correlate these simulated findings with a real Kademlia DHT implementation, figure 7.11 shows the CDF of the total duration of providing 100 sets of 80 CIDs in the IPFS network (from the beginning of the first CID provide operation, to the end of the last one in a single set of 80 CIDs). The figure shows that barely 20% of the hundred sets of provides concluded within 70 seconds, while almost 67% of the provides sets were limited by the timeout of 80 seconds for each provide. When considering the network load of seeding the DHT, it is important to remember that block propagation is also done to validators using GossipSub. However, in that case, instead of sending individual samples to selected target nodes, the efficient mesh-based GossipSub protocol is used to send entire rows and columns. As mentioned before, serving sampling requests directly from validators would compromise the identity of individual validator nodes, and a default Kademlia DHT can't handle that level of request in such a short time range. Thus, this raises the question: how do we seed the DHT, and can a DHT still be used as a blob addressing protocol?

7.4.3 DHT limitations

The DHT could be seeded by the block proposer/builder, but also by validators (limited to the scope of their GossipSub subscription). However, we can state that both cases are problematic for several reasons explained in the following paragraphs:

Single seeder to the DHT

If the block proposer/builder is in charge of the DHT seeding, the nodes in its routing table will suffer the biggest workload of replying to the first hops of the 262.144 requests, creating a huge workload that the network can't handle in 12 seconds as figure 7.10 shows. The tedious problem of having to perform many provide operations to the DHT network is already present in the IPFS DHT for large service

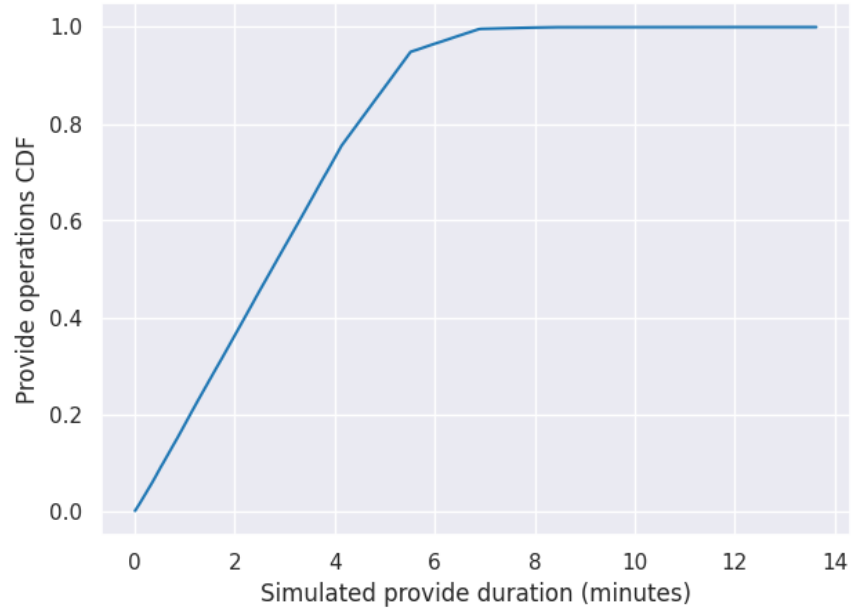

FIGURE 7.10: CDF of the simulated 262.144 concurrent DHT provide operations from a single node.

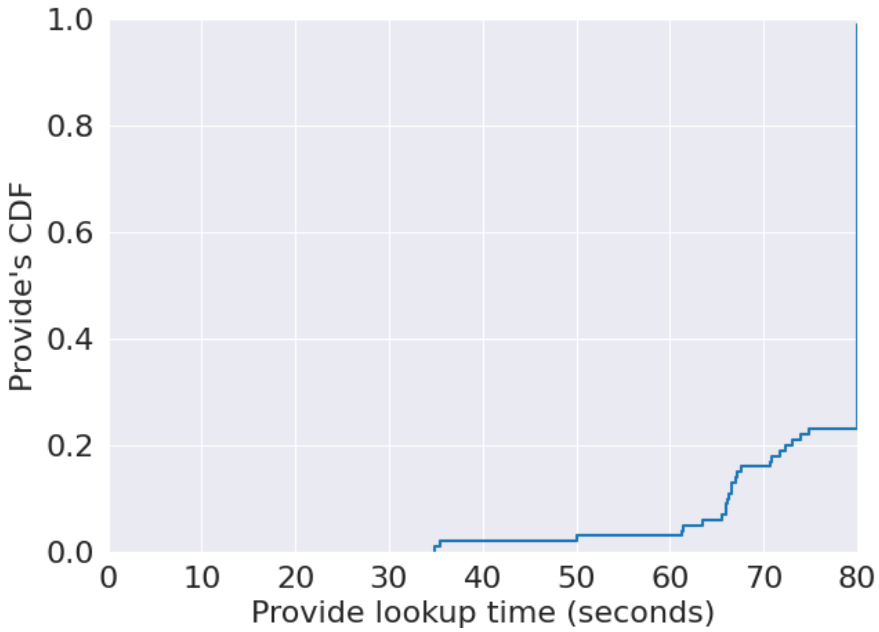

FIGURE 7.11: CDF of the time to finish 100 sets of 80 concurrent CID provides to the IPFS network.

providers like Cloudflare [35] or Pinata [147]. As a result, organizations and developers have moved the biggest part of the content in IPFS away from the DHT, relying on several solutions:

- BitSwap [167], a protocol designed to download content from a connected network participant optimally. Using BitSwap connections as a solution avoids seeding the "provider records" for each hosted item. However, it relies on the nature of the network to end up connected to the largest DHT nodes, which centralizes the network and doesn't provide any content-addressing solution. It relies on the probability of eventually being connected to one of these "big" nodes to retrieve the desired content, not guaranteeing minimal retrieval times for a non-popular item.

- Network Indexers[10]. It is a separate method that tackles the scalability problems of the DHT and the content addressing limitations of BitSwap. It offers a network of high throughput and low latency DHT-like databases present in the network as nodes that can be used by users supporting the protocols. As expected, this solution breaks the nature of distributed networks, as it relies on adding a particular actor whose only function is to provide the content addressing in the network. The solution, despite being scalable, magnifies the exposition of the network to a denial of service attack. Thus, it is not a viable option for Ethereum.

- The Accelerated Routing Table[11]. This experimental routing table enables the possibility of making batch provides or reprovide sweeps[12]. The method relies on the crawling capabilities of the DHT host to discover all the participants in the network. Based on the whole pack of content it has to advertise to the network, it can dismiss every DHT lookup to discover the K closest peers as it already knows the entire list of participants. As a result, a node using an accelerated routing table can perform a bulk-provide operation per each node it needs to connect. The method saves many round-trip connections but exposes the network to the feared node churn. One could think about increasing the frequency of refreshing the routing table to overcome the node churn; however, it would also mean crawling the network more often, replacing the network's spam from DHT lookups and refreshing the accelerated routing table. Furthermore, the Ethereum network tends to have a low level of connections to limit the bandwidth usage [38], which makes it unviable to implement at the moment.

- Recursive (and forwarding) Kademlia. The cost of seeding a large number of small segments (key-value pairs) into a Kademlia DHT is expensive due to the large number of interactions needed from the iterative lookup process, the parallel lookups (alpha), and the number of target storage nodes (K). [78] introduced recursive Kademlia to reduce the iterative overhead, while [41, 168] extended this to support efficient broadcasting in the Kademlia address space. While DAS seeding is more complex than broadcasting, and recursive forwarding introduces compromises in the robustness of seeding the DHT; similar solutions can be considered to reduce the network load and time required.

[10] https://github.com/ipni
[11] https://github.com/ipfs/kubo/pull/8997/files
[12] https://github.com/libp2p/go-libp2p-kad-dht/issues/824

Multiple seeders to the DHT

On the other hand, the seeding could be done more organically by peers subscribed to a small set of GossipSub topics. This would balance the workload of publishing many items to the DHT for individual routing tables of the seeding nodes. But even with more sophisticated solutions like Optimistic Provide [195], the network ends up over-flooded by the same message as more nodes perform the same provide operation. This solution could be improved by adding some logic to the consensus, where validators rely on deterministic randomness to define who must seed the DHT. This solution could follow a similar approach to the one that defines which validator aggregates and distributes the attestations at each slot committee. However, it requires a heavy change in the current logic of the consensus and could expose the privacy of the validators to some degree.

Avoiding the DHT

Finally, we should mention that while writing this chapter, some early-stage emerging proposals try to avoid using a DHT for sampling. PeerDAS [144] relies on a deterministic segment-to-nodeID mapping scheme to avoid expensive lookups. Sampling nodes are assumed to know enough node IDs to be able to perform their sampling based on the mapping. While a peer sampling service is assumed, the compromise of PeerDAS compared to the DHT-based solution is a reduced number of rows and columns. Another proposal, SubnetDAS [186], instead compromises on the randomness of the sampling, more specifically on the unlinkability of the queries. While in a DHT-based design, every single node can query its selected random samples, in SubnetDAS, nodes subscribe to GossipSub channels and hold these subscriptions for a specified period, weakening the protection of nodes against double-spend.

7.5 Conclusion

This chapter presented the results and limitations of implementing a direct Kademlia DHT as a DAS solution for Ethereum's scalability proposal. We offer a DHT simulator that is validated using experiments in a production p2p network with an actual Kademlia DHT implementation. Our results expose that, despite being the right fit for data sampling, seeding the DHT in the context of Ethereum DAS is problematic. We discuss several possible alternative ways to seed the DHT, as elements of other designs not relying on a DHT. In future work, we intend to explore the viability and performance of DHT solutions in the context of more flexible and less constrained DAS protocols.

Chapter 8

Conclusions

There is no doubt that peer-to-peer networks have had a significant impact on blockchain's application layer performance. It is also clear that decentralized computing will be at the core of these advancements as networks and transaction volumes increase. The work carried out in this thesis is conclusive; it has led to key methodologies, tools, and findings that substantially help understand the impact of the different involved peer-to-peer network parameters in the performance of blockchain applications. In this chapter, we first summarize the chapters in this thesis and the conclusions from each chapter. We then provide key takeaways and lessons from this thesis and finally present ideas that couldn't be explored due to time limitations but will be pursued as future work.

8.1 Summary of Achievements

Chapter 1 presented the journey of peer-to-peer networks since the early 2000s and their role in the future of distributed blockchain and web3 applications and protocols. We motivated the study of the topology of a recently upgraded Ethereum network, which adopted the Libp2p network protocol stack for its transition from PoW to PoS. We then used this motivation to present the challenges and impact of the underlying peer-to-peer network on the application layers that run on top of them. Such challenges include the limitations the selected hardware, software, and connective might have on the consensus performance, as well as the limitations that the existing protocols and topology might imply to the scalability of these demanding applications. We also shared a summary of the objectives and contributions of this thesis.

In chapter 2, we gave a background of the current state of the art in the evolution of peer-to-peer networks, focusing later on Ethereum's upgrade towards PoS. We also explained how Ethereum works and the different protocols, terms, and networks used in this thesis to conduct experiments. Though related work is provided in each chapter, we have a short related work for Ethereum's PoS consensus and the main components of a peer-to-peer network.

Chapter 3 described the internal topology of Ethereum's PoS peer-to-peer network. We've provided a detailed representation of the health of the network, providing relevant information such as the number of advertised Ethereum nodes, the number of reachable nodes from those advertised nodes, the geographical location of the same ones, the software they are using, the latency between our tool and the respective nodes among others. The chapter presents the clusterization or centralization level that the network is subjected to, as it describes the big concentration of nodes in only two countries, the distribution of nodes that share the same IP, or the nodes that are hosted in data centres. The study highlights that the centralization vectors that the network currently suffers can directly impact the performance of

the application layer, presenting, furthermore, the challenges that the network had to overcome to ensure a certain resilience degree at the networking layer. We also shared a summary of the objectives and contributions of this thesis.

In chapter 4, we detailed the available software and hardware solutions to participate in Ethereum's network. The chapter introduces how the transition from PoW to PoS targeted a reduction in the hardware requirements, enabling more users' participation in the network and, thus, the consensus. The conducted experiments showcase how each client performs on different hardware conditions, highlighting the main difference between clients and hardware platforms. Furthermore, the chapter provides a performance analysis comparing the different states that a node has while syncing the chain and under different workloads. The Chapter summarises the insights to help users define which software and platform they should choose for their specific use case.

Chapter 5 presents the game theory behind the incentives that the new PoS promotes to stimulate honest behaviour among active validators. We introduce a novel methodology that measures the performance of any active validator at any period, which can be used to compare any set of validators evenly later. The chapter introduces a detailed description of how the rewards are calculated at the consensus layer. It describes how we can use the consensus specifications to recompute the reward shares that each duty has generated over some time. Furthermore, the chapter analyzes the changes in the consensus participants that "The Merge" of the execution layer and the beacon chain meant for the ecosystem. It introduces how, despite having a larger ratio of decentralization among active validators and staking organizations, PoS gives these entities bigger chances to identify the production of consecutive blocks.

Chapter 6 is the core chapter of this thesis. This chapter's work was conducted over almost two years; hence, it's the longest chapter. This chapter first introduced direct implications that the peer-to-peer network overlay might infer on the application layer, where being "far" from the network's core in the long term produces less profitability to active validators. To some degree, we identified that the short time window inside each slot defined by the consensus can incentivise users and entities to cluster to minimize the latency geographically. We then extrapolate this funding to test the viability of the promising Distributed Validator Technology, where we could demonstrate that if the DVT participants have low latencies to themself and if the hardware is well dimensioned, it can match and even, in some cases, improve the performance and profitability of any canonical validator. Finally, we provide a summary of the analysis.

Chapter 7 presents the content addressing challenges that Ethereum's scalability is facing with the apparition of blob transactions with *EIP-4844* and the proposal of *DankSharding*. The chapter presents one of the most resounding options to achieve it, a Kademlia DHT, showcasing the existing limitations and bottlenecks that this one would mean to the protocol due to the high expected throughput. The chapter presents an open-source simulator that can test different network conditions. We present and validate the simulation using IPFS's public DHT, highlighting the arduous process of seeding a DHT with blob samples. The chapter compiles a set of partial solutions that could speed up the DHT seeding process, finalizing the chapter with the biggest counterparts that they have: a high demand from the network nodes.

8.2 Notable Observations

In the introduction, we embarked on a journey to study the topology and the impact of the modern peer-to-peer network on such time-sensitive applications as blockchain and the web3 space. This section highlights some of the key observations and findings from this thesis.

Centralization vectors: The transition from Pow to PoS meant reducing hardware requirements to participate in consensus, significantly impacting the network. It didn't only define specific time ranges for each duty within a slot; we spotted how those narrow ranges affect the performance of validators and influence how the protocol further evolves. We've measured an increment in the total number of nodes actively participating in the network, bringing resilience to the same one. However, the concentration of nodes in two major countries and two major software and the concentration of a third of the active validators in a single staking entity raise some centralization warnings.

Validator Duties as Performance Indicators: As opposed to the previous PoW consensus, Ethereum's PoS clearly defines the duties each validator would have to accomplish and the reward or personalisation it should get. Such transparency allows us to directly determine the performance and profitability of a validator based on their accomplishments. We've presented a novel methodology with indicators and scores that enable us to judge how well a validator behaves. This provides insights into the leading causes that could originate any behaviour behind the expected one. The methodology offers a fair baseline to compare validators, enabling the comparison of staking entities, distinct available software, or any networking alteration.

Strict Time Ranges Impose Limitations into Validators' Performance and Future Protocol Upgrades: Defining such strict time guidelines simplifies and eases the consensus across the validators. It splits the workload of processing a high number of attestations per epoch across multiple slots and committees. However, with the apparition of extra-protocolary processes such as MEV or DVT, the proposal of new blocks has been somewhat delayed. On some occasions, this has compromised the capability of some validators to accomplish such vital duties, highlighting that the four-second window to generate and propagate duties might start to be a real bottleneck. This is even more relatable when testing the proposed scalability solutions, which suffer from having to share a significantly larger number of bytes in such a short period.

8.3 Future Outlooks

In a span of three years of hard work, limitations are inevitable. Decisions regarding priorities and the depth of research needed to be carefully considered. As such, the conclusion of this thesis does not signify the cessation of inquiry into the topics discussed herein. Many concepts have been introduced but not explored in depth. These areas will receive further attention in future work. For instance, in chapter 3, we presented a significant discrepancy between the number of Ethereum node records found in the DHT and the number of peers we connected with. However, whether this occurs due to a large number of nodes behind NAT or a misconfiguration of the DHT pruning mechanism to remove stale records is still unclear. Future work will expand the work to prevent any possible eclipse attack over the peer discovery DHT. In chapter 6, we presented how user choices, location, and even

new emerging technologies can impact the performance of a validator. However, a whole analysis like the one presented must be conducted to understand which parts of a node are underperforming. Future work will try to provide a detailed score summarising the state of an Ethereum full node (EL + CL clients) concerning the network. This would allow debugging in a more accessible way, which is the node parameter bottlenecking the whole deployment.

Appendix A

Publications and Dissemination

A.1 Publications

1. **M. Cortes-Goicoechea** and L. Bautista-Gomez, "Discovering the Ethereum2 P2P Network," 2021 Third International Conference on Blockchain Computing and Applications (BCCA), Tartu, Estonia, 2021, pp. 81-88, doi: 10.1109/BCCA53669.2021.9657041.

2. **M. Cortes-Goicoechea**, L. Franceschini and L. Bautista-Gomez, "Resource Analysis of Ethereum 2.0 Clients," 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 2021, pp. 1-8, doi: 10.1109/BRAINS52497.2021.9569812.

3. **M. Cortes-Goicoechea**, T. Mohandas-Daryanani, J. L. Muñoz-Tapia and L. Bautista-Gomez. "Can we run our Ethereum nodes at home?". IEEE Access. doi: 10.1109/ACCESS.2024.3381782.

4. **M. Cortes-Goicoechea**, T. Mohandas-Daryanani, J. L. Muñoz-Tapia and L. Bautista-Gomez, "Autopsy of Ethereum's Post-Merge Reward System," 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Dubai, United Arab Emirates, 2023, pp. 1-9, doi: 10.1109/ICBC56567.2023.10174942.

5. **M. Cortes-Goicoechea**, T. Mohandas-Daryanani, J. L. Muñoz-Tapia and L. Bautista-Gomez, "Unveiling Ethereum's Hidden Centralization Incentives: Does Connectivity Impact Performance?," 2023 Fifth International Conference on Blockchain Computing and Applications (BCCA), Kuwait, Kuwait, 2023, pp. 89-96, doi: 10.1109/BCCA58897.2023.10338892.

6. **M. Cortes-Goicoechea**, T. Mohandas-Daryanani, J. L. Muñoz-Tapia and L. Bautista-Gomez. "DVT in Ethereum's PoS: Gains and Loses" Under submission process at "Elsevier: Computer-Communication".

7. P. Ocheja, **M. Cortes-Goicoechea**, T. Mohandas-Daryanani, and L. Bautista-Gomez. An Analytical Study of Large Blocks on Ethereum. TO BE PUBLISHED ON "The 2024 6th Blockchain and Internet of Things Conference" Fukuoka, Japan, 2024

8. **Cortes-Goicoechea M**, Kiraly C, Ryajov D, Muñoz-Tapia JL, Bautista-Gomez L. Scalability limitations of Kademlia DHTs when enabling Data Availability Sampling in Ethereum. arXiv preprint arXiv:2402.09993. 2024 Feb 15. TO BE PUBLISHED ON "The 2024 6th Blockchain and Internet of Things Conference" Fukuoka, Japan, 2024

A.2 Workshops and Talks

1. **Provider Records: do they stay alive long enough? IPFS þhing 2022 Reykjavík, Iceland**
2. **A Deep Dive into Provider Records, IPFS Camp 2022, Lisbon, Portugal**

A.3 Research Collaboration

1. **Codex Storage, Status Im, Switzerland, 2023**
   Three months of research collaboration with the Codex Storage team working on the content addressing the challenges of upgrading Ethereum's scalability.
2. **Cambridge Center for Alternative Finance, Cambridge, UK, 2023**
   Two months of research stay at the Cambridge Center for Alternative Finance from the Cambridge Judge Business School department, working on modelling the energy consumption of Ethereum's network.

A.4 Grants

1. **Protocol Labs Research PhD Fellowship**
2. **Ethereum Foundation's Devcon IV Scholars Program**

Appendix B

Provider Record's Liveness in IPFS

As a result of the research grant and the collaboration with Protocol Labs and the ProbeLab team, over the course of the thesis, we had the chance to perform two research reports to the IPFS DHT. These works relate to the healthiness of the *Provider Records* in the IPFS DHT.

In the first one [118], we describe a fully open-source tool that can track how many of the redundant records stored in the DHT remain available over the protocol-defined 24 hours, the connectivity of the nodes initially contacted to keep the records, and the accuracy of these initial nodes over the lifetime of the records. The report highlights the correct performance of the network, suggesting some actions that could reduce the overhead of the network and certain centralization of the same one without directly impacting the performance.

As an extension of the first report, the second one [119] presents how the protocol, due to some implementation limitations, forces users to perform two separate lookups to retrieve any content in the network. The report presents how this second lookup could be omitted by extending the Time To live (TTL) of each content provider's multiaddresses to match the TTL of the provider records. This way the records could be shared alongside the multiaddresses straight away on a single lookup.

Bibliography

[1] Michael Abd-El-Malek et al. "Fault-scalable Byzantine fault-tolerant services". In: *ACM SIGOPS Operating Systems Review* 39.5 (2005), pp. 59–74.

[2] Zeyad A Al-Odat et al. "Secure hash algorithms and the corresponding FPGA optimization techniques". In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–36.

[3] Bithin Alangot et al. "Decentralized and lightweight approach to detect eclipse attacks on proof of work blockchains". In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1659–1672.

[4] Elvira Albert et al. "GASOL: gas analysis and optimization for ethereum smart contracts". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2020, pp. 118–125.

[5] *API Benchmark tool*. URL: https://github.com/cortze/api-benchmark.

[6] *Armiarma Crawler*. https://github.com/migalabs/armiarma.

[7] Rameez Asif and Syed Raheel Hassan. "Shaping the future of Ethereum: Exploring energy consumption in Proof-of-Work and Proof-of-Stake consensus". In: *Frontiers in Blockchain* 6 (2023).

[8] Attestant.io. *Vouch*. https://github.com/attestantio/vouch. 2020.

[9] José Eduardo de Azevedo Sousa et al. "An analysis of the fees and pending time correlation in Ethereum". In: *International Journal of Network Management* 31.3 (2021), e2113.

[10] François Baccelli et al. "Can P2P networks be super-scalable?" In: *2013 Proceedings IEEE INFOCOM*. IEEE. 2013, pp. 1753–1761.

[11] Foteini Baldimtsi et al. "Subset-optimized BLS Multi-signature with Key Aggregation". In: *Cryptology ePrint Archive* (2023).

[12] Leonhard Balduf et al. "Monitoring data requests in decentralized data storage systems: A case study of IPFS". In: *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2022, pp. 658–668.

[13] Shane Balfe, Amit D Lakhani, and Kenneth G Paterson. "Trusted computing: Providing security for peer-to-peer networks". In: *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*. IEEE. 2005, pp. 117–124.

[14] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. "A survey of blockchain consensus algorithms performance evaluation criteria". In: *Expert Systems with Applications* 154 (2020), p. 113385.

[15] Guruduth Banavar et al. "An efficient multicast protocol for content-based publish-subscribe systems". In: *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No. 99CB37003)*. IEEE. 1999, pp. 262–272.

[16] Sami Ben Mariem, Pedro Casas, and Benoît Donnet. "Vivisecting blockchain P2P networks: Unveiling the Bitcoin IP network". In: *ACM CoNEXT student workshop*. 2018.

[17] Juan Benet. "Ipfs-content addressed, versioned, p2p file system". In: *arXiv preprint arXiv:1407.3561* (2014).

[18] Grace J Bergen. "The Napster Case: The Whole World is Listening". In: *Transnat'l LAW.* 15 (2002), p. 259.

[19] Wei Bi, Xiaoyun Jia, and Maolin Zheng. "A secure multiple elliptic curves digital signature algorithm for blockchain". In: *arXiv preprint arXiv:1808.02988* (2018).

[20] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. "Deanonymisation of clients in Bitcoin P2P network". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 15–29.

[21] Rainer Böhme et al. "Bitcoin: Economics, technology, and governance". In: *Journal of economic Perspectives* 29.2 (2015), pp. 213–238.

[22] Dan Boneh, Manu Drijvers, and Gregory Neven. "Bls multi-signatures with public-key aggregation". In: *URL: https://crypto. stanford. edu/˜ dabo/pubs/papers/BLSmultisig. html* (2018).

[23] Daniel Borbor et al. "Optimizing the network diversity to improve the resilience of networks against unknown attacks". In: *Computer Communications* 145 (2019), pp. 96–112.

[24] Barry Bozeman and Elizabeth Corley. "Scientists' collaboration strategies: implications for scientific and technical human capital". In: *Research policy* 33.4 (2004), pp. 599–616.

[25] Vitalik Buterin and Virgil Griffith. "Casper the friendly finality gadget". In: *arXiv preprint arXiv:1710.09437* (2017).

[26] Vitalik Buterin et al. "Combining GHOST and casper". In: *arXiv preprint arXiv:2003.03052* (2020).

[27] Wei Cai et al. "Decentralized applications: The blockchain-empowered software system". In: *IEEE access* 6 (2018), pp. 53019–53033.

[28] Bengt Carlsson and Rune Gustavsson. "The rise and fall of napster-an evolutionary approach". In: *International Computer Science Conference on Active Media Technology*. Springer. 2001, pp. 347–354.

[29] Franck Cassez, Joanne Fuller, and Aditya Asgaonkar. "Formal verification of the ethereum 2.0 beacon chain". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2022, pp. 167–182.

[30] Miguel Castro, Barbara Liskov, et al. "Practical byzantine fault tolerance". In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186.

[31] Miguel Castro et al. "Exploiting network proximity in distributed hash tables". In: *International Workshop on Future Directions in Distributed Computing (FuDiCo)*. Citeseer. 2002, pp. 52–55.

[32] *Charon*. https://github.com/ObolNetwork/charon.

[33] Anamika Chauhan et al. "Blockchain and scalability". In: *2018 IEEE international conference on software quality, reliability and security companion (QRS-C)*. IEEE. 2018, pp. 122–128.

[34] Huashan Chen et al. "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses". In: *ACM Computing Surveys (CSUR)* 53.3 (2020), pp. 1–43.

[35] *Cloudflare*. URL: https://www.cloudflare.com/. (accessed: 08.02.2024).

[36] Mikel Cortes-Goicoechea. *IPFS Provider Record's Liveness*. URL: https://github.com/plprobelab/network-measurements/blob/master/results/rfm17-provider-record-liveness.md. (accessed: 08.02.2024).

[37] Mikel Cortes-Goicoechea and Leonardo Bautista-Gomez. "Discovering the Ethereum2 P2P network". In: *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*. IEEE. 2021, pp. 81–88.

[38] Mikel Cortes-Goicoechea, Luca Franceschini, and Leonardo Bautista-Gomez. "Resource analysis of Ethereum 2.0 clients". In: *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE. 2021, pp. 1–8.

[39] Mikel Cortes Goicoechea et al. *Miga Labs' Ethereum Client's hardware resource analysis*. URL: https://github.com/migalabs/eth-client-hw-analysis.

[40] Corinne Curt and Jean-Marc Tacnet. "Resilience of critical infrastructures: Review and analysis of current approaches". In: *Risk Analysis* 38.11 (2018), pp. 2441–2458.

[41] Zoltán Czirkos and Gábor Hosszú. "Solution for the broadcasting in the Kademlia peer-to-peer overlay". In: *Computer Networks* 57.8 (2013), pp. 1853–1862. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2013.02.021. URL: https://www.sciencedirect.com/science/article/pii/S1389128613000777.

[42] Peter Dahlgren. "The Internet and the democratization of civic culture". In: *Political communication* 17.4 (2000), pp. 335–340.

[43] Philip Daian et al. "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges". In: *arXiv preprint arXiv:1904.05234* (2019).

[44] Erik Daniel and Florian Tschorsch. "Passively Measuring IPFS Churn and Network Size". In: *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE. 2022, pp. 60–65.

[45] *DankSharding*. URL: https://ethereum.org/en/roadmap/danksharding/. (accessed: 08.02.2024).

[46] *Data availability sampling and danksharding: An overview and a proposal for improvements*. URL: https://a16zcrypto.com/posts/article/an-overview-of-danksharding-and-a-proposal-for-improvement-of-das/. (accessed: 08.02.2024).

[47] Sergi Delgado-Segura et al. "Cryptocurrency networks: A new P2P paradigm". In: *Mobile Information Systems* 2018 (2018).

[48] Valeriia Denisova. "Blockchain infrastructure and growth of global power consumption". In: *International Journal of Energy Economics and Policy* (2019).

[49] Varun Deshpande, Hakim Badis, and Laurent George. "Btcmap: Mapping bitcoin peer-to-peer network topology". In: *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. IEEE. 2018, pp. 1–6.

[50] Jega Anish Dev. "Bitcoin mining acceleration and performance quantification". In: *2014 IEEE 27th Canadian conference on electrical and computer engineering (CCECE)*. IEEE. 2014, pp. 1–6.

[51] *DHT simulations on Codex-DAS research*. URL: https://github.com/codex-storage/das-research/pull/52. (accessed: 08.02.2024).

[52] *Discovery5 Protocol*. URL: https://github.com/ethereum/devp2p/blob/master/discv5/discv5.md. (accessed: 08.03.2024).

[53] Jack Dongarra, Thomas Herault, and Yves Robert. *Fault tolerance techniques for high-performance computing*. Springer, 2015.

[54] Hendrik Drachsler and Wolfgang Greller. "Privacy and analytics: it's a DELICATE issue a checklist for trusted learning analytics". In: *Proceedings of the sixth international conference on learning analytics & knowledge*. 2016, pp. 89–98.

[55] Kevin Driscoll et al. "Byzantine fault tolerance, from theory to reality". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2003, pp. 235–248.

[56] *EIP-4844*. URL: https://eips.ethereum.org/EIPS/eip-4844#blob-transaction. (accessed: 08.02.2024).

[57] *Eth2 Mainnet Incident Retrospective*. https://medium.com/prysmatic-labs/eth2-mainnet-incident-retrospective-f0338814340c.

[58] *Ethereum Beacon Node REST API Standard*. https://ethereum.github.io/beacon-APIs/.

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

[60] *Ethereum Foundation*. https://ethereum.foundation/.

[61] *Ethereum Node Records (ENR)*. URL: https://eips.ethereum.org/EIPS/eip-778. (accessed: 08.03.2024).

[62] *Ethereum whitepaper*. https://ethereum.org/en/whitepaper/.

[63] *Ethereum's Consensus Specs*. https://github.com/ethereum/consensus-specs.

[64] *Ethereum's RNG RANDAO*. https://github.com/randao/randao.

[65] *Etherscan, Ethereum Blockchain Explorer*. https://etherscan.io/.

[66] Paola Flocchini, Amiya Nayak, and Ming Xie. "Enhancing peer-to-peer systems through redundancy". In: *IEEE Journal on selected areas in communications* 25.1 (2007), pp. 15–24.

[67] Geoffrey Fox. "Peer-to-peer networks". In: *Computing in Science & Engineering* 3.3 (2001), pp. 75–77.

[68] Jianbin Gao et al. "Supply chain equilibrium on a game theory-incentivized blockchain network". In: *Journal of Industrial Information Integration* 26 (2022), p. 100288.

[69] Yue Gao et al. "Topology Measurement and Analysis on Ethereum P2P Network". In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2019, pp. 1–7.

[70] *General Specs of the dv5 protocol*. https://github.com/ethereum/devp2p/blob/master/discv5/discv5.md.

[71] Arthur Gervais et al. "On the security and performance of proof of work blockchains". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 3–16.

[72] Mark Graham and William H Dutton. *Society and the internet: How networks of information and communication are changing our lives*. Oxford University Press, 2019.

[73] Jens Groth. "Non-interactive distributed key generation and key resharing". In: *Cryptology ePrint Archive* (2021).

[74] Kobi Gurkan et al. "Aggregatable distributed key generation". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pp. 147–176.

[75] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. "Foundations of Data Availability Sampling". In: *Cryptology ePrint Archive* (2023).

[76] Adam Hayes. "A cost of production model for bitcoin". In: *Available at SSRN 2580904* (2015).

[77] Zhiguo He, Jiasun Li, and Zhengxun Wu. "Don't Trust, Verify: The Case of Slashing from a Popular Ethereum Explorer". In: *Companion Proceedings of the ACM Web Conference 2023*. 2023, pp. 1078–1084.

[78] Bernhard Heep. "R/Kademlia: Recursive and topology-aware overlay routing". In: *2010 Australasian Telecommunication Networks and Applications Conference*. IEEE. 2010, pp. 102–107.

[79] Sebastian Henningsen et al. "Crawling the IPFS network". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 679–680.

[80] Everett Hildenbrandt et al. "Kevm: A complete formal semantics of the ethereum virtual machine". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, pp. 204–217.

[81] Joanne Hinds, Emma J Williams, and Adam N Joinson. ""It wouldn't happen to me": Privacy concerns and perspectives following the Cambridge Analytica scandal". In: *International Journal of Human-Computer Studies* 143 (2020), p. 102498.

[82] Yoichi Hirai. "Defining the ethereum virtual machine for interactive theorem provers". In: *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer. 2017, pp. 520–535.

[83] Nicolas Houy. "The bitcoin mining game". In: *Available at SSRN 2407834* (2014).

[84] Andrew Howell, Takfarinas Saber, and Malika Bendechache. "Measuring node decentralisation in blockchain peer to peer networks". In: *Blockchain: Research and Applications* 4.1 (2023), p. 100109.

[85] Yuming Huang et al. "Do the rich get richer? Fairness analysis for blockchain incentives". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 790–803.

[86] Chris Huxham and Siv Vangen. *Managing to collaborate: The theory and practice of collaborative advantage*. Routledge, 2013.

[87] Muhammad Anas Imtiaz et al. "Churn in the bitcoin network: Characterization and impact". In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 431–439.

[88] Tatsushi Inagaki et al. "Profile-based detection of layered bottlenecks". In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. 2019, pp. 197–208.

[89] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* 1 (2001), pp. 36–63.

[90] Mihajlo Jovanović, Fred Annexstein, and Kenneth Berman. "Modeling peer-to-peer network topologies through small-world models and power laws". In: *IX Telecommunications Forum, TELFOR*. Citeseer. 2001, pp. 1–4.

[91] Ben Kaiser, Mireya Jurado, and Alex Ledger. "The looming threat of china: An analysis of chinese influence on bitcoin". In: *arXiv preprint arXiv:1810.02466* (2018).

[92] Harry Kalodner et al. "Arbitrum: Scalable, private smart contracts". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 1353–1370.

[93] Elie Kapengut and Bruce Mizrach. "An event study of the ethereum transition to proof-of-stake". In: *Commodities* 2.2 (2023), pp. 96–110.

[94] Aniket Kate, Yizhou Huang, and Ian Goldberg. "Distributed key generation in the wild". In: *Cryptology ePrint Archive* (2012).

[95] Ali Khajeh-Hosseini, David Greenwood, and Ian Sommerville. "Cloud migration: A case study of migrating an enterprise it system to iaas". In: *2010 IEEE 3rd International Conference on cloud computing*. IEEE. 2010, pp. 450–457.

[96] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[97] Lucianna Kiffer et al. "Under the Hood of the Ethereum Gossip Protocol". In: *Proceedings of the 2021 International Conference on Financial Cryptography and Data Security (FC'21)*. 2021.

[98] MinJi Kim et al. "On counteracting byzantine attacks in network coded peer-to-peer networks". In: *IEEE Journal on Selected Areas in Communications* 28.5 (2010), pp. 692–702.

[99] Seoung Kyun Kim et al. "Measuring ethereum network peers". In: *Proceedings of the Internet Measurement Conference 2018*. 2018, pp. 91–104.

[100] Ramayya Krishnan, Michael D Smith, and Rahul Telang. "The economics of peer-to-peer networks". In: *Available at SSRN 504062* (2003).

[101] Bartosz Kusmierz and Roman Overko. "How centralized is decentralized? Comparison of wealth distribution in coins and tokens". In: *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE. 2022, pp. 1–6.

[102] James SH Kwok and Sheng Gao. "Knowledge sharing community in P2P network: a study of motivational perspective". In: *Journal of knowledge Management* 8.1 (2004), pp. 94–102.

[103] Kin-Wah Kwong and Danny HK Tsang. "Building heterogeneous peer-to-peer networks: protocol and analysis". In: *IEEE/ACM transactions on networking* 16.2 (2008), pp. 281–292.

[104] *KZG polynomial commitments*. URL: https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html. (accessed: 08.02.2024).

[105] Bahareh Lashkari and Petr Musilek. "A comprehensive review of blockchain consensus mechanisms". In: *IEEE Access* 9 (2021), pp. 43620–43652.

[106] Elaine Lawrence, John Lawrence, and Gordana Culjak. "Legal and technical issues management framework for peer-to-peer networks". In: *Journal of Theoretical and Applied Electronic Commerce Research* 1.1 (2006), pp. 32–41.

[107] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. "Understanding bittorrent: An experimental perspective". In: (2005).

[108] Barry M Leiner et al. "A brief history of the Internet". In: *ACM SIGCOMM computer communication review* 39.5 (2009), pp. 22–31.

[109] Derek Leonard, Vivek Rai, and Dmitri Loguinov. "On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks". In: *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 2005, pp. 26–37.

[110] Jingming Li et al. "Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies". In: *Energy* 168 (2019), pp. 160–168.

[111] Kai Li, Yibo Wang, and Yuzhe Tang. "Deter: Denial of ethereum txpool services". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 1645–1667.

[112] Yehuda Lindell and Ariel Nof. "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1837–1854.

[113] Alexander Lipton. "Blockchains and distributed ledgers in retrospective and perspective". In: *The Journal of Risk Finance* 19.1 (2018), pp. 4–25.

[114] Bev Littlewood, Peter Popov, and Lorenzo Strigini. "Modeling software design diversity: a review". In: *ACM Computing Surveys (CSUR)* 33.2 (2001), pp. 177–208.

[115] Bev Littlewood and Lorenzo Strigini. "Redundancy and diversity in security". In: *Computer Security–ESORICS 2004: 9th European Symposium on Research in Computer Security, Sophia Antipolis, France, September 13-15, 2004. Proceedings 9*. Springer. 2004, pp. 423–438.

[116] Ziyao Liu et al. "A survey on applications of game theory in blockchain". In: *arXiv preprint arXiv:1902.10865* (2019).

[117] *Looking up IPFS*. URL: https://github.com/cortze/looking-up-ipfs. (accessed: 08.02.2024).

[118] L. Bautista-Gomez M. Cortes-Goicoechea. *RFM 17 | Provider Record Liveness*. URL: https://github.com/probe-lab/network-measurements/blob/main/results/rfm17-provider-record-liveness.md. (data: 18.08.2022, accessed: 08.03.2024).

[119] L. Bautista-Gomez M. Cortes-Goicoechea. *RFM 17.1 | Sharing Provider Records with Multiaddresses*. URL: https://github.com/probe-lab/network-measurements/blob/main/results/rfm17.1-sharing-prs-with-multiaddresses.md. (data: 11.11.2022, accessed: 08.03.2024).

[120] Azad M Madni and Scott Jackson. "Towards a conceptual framework for resilience engineering". In: *IEEE Systems Journal* 3.2 (2009), pp. 181–191.

[121] Aidan O Mahony and Emanuel Popovici. "A systematic review of blockchain hardware acceleration architectures". In: *2019 30th Irish signals and systems conference (ISSC)*. IEEE. 2019, pp. 1–6.

[122] Ripeanu Matei, Adriana Iamnitchi, and P Foster. "Mapping the Gnutella network". In: *IEEE Internet Computing* 6.1 (2002), pp. 50–57.

[123] Petar Maymounkov and David Mazieres. "Kademlia: A peer-to-peer information system based on the xor metric". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65.

[124] *Medalla non-finality period August 2020:* https://docs.google.com/document/d/11RmitNRui10LcLCyoXY6B1INCZZKq30gEU6BEg3EWfk.

[125] MigaLabs. *Ethereum Consensus Layer's State Analyzer*. https://github.com/migalabs/goteth. 2022.

[126] Andrew Miller. "Permissioned and permissionless blockchains". In: *Blockchain for distributed systems security* (2019), pp. 193–204.

[127] Tarun Mohandas-Daryanani. *Ethereum Consensus Layer's Block Scorer*. https://github.com/migalabs/eth-cl-live-metrics. 2022.

[128] Bhabendu Kumar Mohanta et al. "Blockchain technology: A survey on applications and security privacy challenges". In: *Internet of Things* 8 (2019), p. 100107.

[129] Olivier Moindrot and Charles Bournhonesque. "Proof of stake made simple with casper". In: *ICME, Stanford University* (2017).

[130] Bruce Momjian. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York, 2001.

[131] Satoshi Nakamoto. "Bitcoin whitepaper". In: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* 9 (2008), p. 15.

[132] Satoshi Nakamoto et al. "Bitcoin". In: *A peer-to-peer electronic cash system* 21260 (2009).

[133] Nethermind.io. *Nethermind*. https://github.com/NethermindEth/nethermind. 2018.

[134] Till Neudecker and Hannes Hartenstein. "Short paper: An empirical analysis of blockchain forks in bitcoin". In: *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer. 2019, pp. 84–92.

[135] Anh Nguyen-Tuong et al. "Security through redundant data diversity". In: *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE. 2008, pp. 187–196.

[136] *Node Exporter*. https://github.com/prometheus/node_exporter.

[137] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. "Monitoring energy hotspots in software: Energy profiling of software code". In: *Automated Software Engineering* 22 (2015), pp. 291–332.

[138] Pedro Henrique FS Oliveira et al. "Analysis of account behaviors in Ethereum during an economic impact event". In: *arXiv preprint arXiv:2206.11846* (2022).

[139] *On-chain scaling to potentially 500 tx/sec through mass tx validation*. URL: https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477. (accessed: 08.02.2024).

[140] A Pinar Ozisik, George Bissias, and Brian Neil Levine. "Estimation of miner hash rates and consensus on blockchains". In: *arXiv preprint arXiv:1707.00082* (2017).

[141] Federica Paganelli, David Parlanti, et al. "A DHT-based discovery service for the Internet of Things". In: *Journal of Computer Networks and Communications* 2012 (2012).

[142] Nikolaos Papadis and Leandros Tassiulas. "Blockchain-based payment channel networks: Challenges and recent advances". In: *IEEE Access* 8 (2020), pp. 227596–227609.

[143] Remigijus Paulavičius et al. "A decade of blockchain: Review of the current status, challenges, and future directions". In: *Informatica* 30.4 (2019), pp. 729–748.

[144] *PeerDAS proposal*. URL: https://ethresear.ch/t/peerdas-a-simpler-das-approach-using-battle-tested-p2p-components/16541. (accessed: 08.02.2024).

[145] Li Peng et al. "Privacy preservation in permissionless blockchain: A survey". In: *Digital Communications and Networks* 7.3 (2021), pp. 295–307.

[146] *Phase 0 – The Beacon Chain*. URL: https://ethereum.github.io/consensus-specs/specs/phase0/beacon-chain/. (accessed: 08.03.2024).

[147] *Pinata*. URL: https://www.pinata.cloud/. (accessed: 08.02.2024).

[148] Giuseppe Pirrò, Domenico Talia, and Paolo Trunfio. "A DHT-based semantic overlay network for service discovery". In: *Future Generation Computer Systems* 28.4 (2012), pp. 689–707.

[149] Giulia Pititto. "The Gasper Protocol: a Proof of Stake Era for Ethereum". PhD thesis. Politecnico di Torino, 2022.

[150] B Pourebrahimi, K Bertels, and S Vassiliadis. "A survey of peer-to-peer networks". In: *Proceedings of the 16th annual workshop on Circuits, Systems and Signal Processing*. 2005, pp. 570–577.

[151] Johan Pouwelse et al. "The bittorrent p2p file-sharing system: Measurements and analysis". In: *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005. Revised Selected Papers 4*. Springer. 2005, pp. 205–216.

[152] *Probelab.io, IPFS lookups' performance*. URL: https://probelab.io/ipfsdht/\#dht-lookup-performance-distribution-by-region. (accessed: 08.02.2024).

[153] *Prometheus*. https://github.com/prometheus/prometheus.

[154] *Proposer-builder separation*. URL: https://ethereum.org/nl/roadmap/pbs/. (accessed: 08.02.2024).

[155] *ProtoDankSharding: EIP-4844*. URL: https://www.eip4844.com/. (accessed: 08.02.2024).

[156] Laura L Pullum. *Software fault tolerance techniques and implementation*. Artech House, 2001.

[157] *py-dht*. URL: https://github.com/cortze/py-dht. (accessed: 08.02.2024).

[158] Kaihua Qin, Liyi Zhou, and Arthur Gervais. "Quantifying blockchain extractable value: How dark is the forest?" In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 198–214.

[159] Zheng Qin. *Introduction to E-commerce*. Springer science & business media, 2010.

[160] J Ramos. "Deep democracy, peer-to-peer production and our common futures". In: *Futura, 31* 4 (2012).

[161] Ahmad Raza, A Rashid Kausar, and David Paul. "The social democratization of knowledge: some critical reflections on e-learning". In: *Multicultural Education & Technology Journal* 1.1 (2007), pp. 64–74.

[162] David Rehak, Pavel Senovsky, and Simona Slivkova. "Resilience of critical infrastructure elements and its main factors". In: *Systems* 6.2 (2018), p. 21.

[163] Pierre Reibel, Haaroon Yousaf, and Sarah Meiklejohn. "Short paper: An exploration of code diversity in the cryptocurrency landscape". In: *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer. 2019, pp. 73–83.

[164] Ana Reyna et al. "On blockchain and its integration with IoT. Challenges and opportunities". In: *Future generation computer systems* 88 (2018), pp. 173–190.

[165] John Riley. "The current status of cryptocurrency regulation in China and its effect around the world". In: *China and WTO Review* 7.1 (2021), pp. 135–152.

[166] Matei Ripeanu. "Peer-to-peer architecture case study: Gnutella network". In: *Proceedings first international conference on peer-to-peer computing*. IEEE. 2001, pp. 99–100.

[167] Alfonso De la Rocha, David Dias, and Yiannis Psaras. "Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin". In: *San Francisco, CA, USA* (2021), p. 11.

[168] Elias Rohrer and Florian Tschorsch. "Kadcast: A Structured Approach to Broadcast in Blockchain Networks". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, 2019, 199–213. ISBN: 9781450367325. DOI: 10.1145/3318041.3355469. URL: https://doi.org/10.1145/3318041.3355469.

[169] Inna Romashkova, Mikhail Komarov, and Aleksandr Ometov. "Demystifying Blockchain Technology for Resource-Constrained IoT Devices: Parameters, Challenges and Future Perspective". In: *IEEE Access* 9 (2021), pp. 129264–129277. DOI: 10.1109/ACCESS.2021.3112228.

[170] Tim Roughgarden. "Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559". In: *arXiv preprint arXiv:2012.00854* (2020).

[171] Sara Rouhani and Ralph Deters. "Performance analysis of ethereum transactions in private blockchain". In: *2017 8th IEEE international conference on software engineering and service science (ICSESS)*. IEEE. 2017, pp. 70–74.

[172] Fahad Saleh. "Blockchain without waste: Proof-of-stake". In: *The Review of financial studies* 34.3 (2021), pp. 1156–1190.

[173] Mehrdad Salimitari et al. "Profit maximization for bitcoin pool mining: A prospect theoretic approach". In: *2017 IEEE 3rd international conference on collaboration and internet computing (CIC)*. IEEE. 2017, pp. 267–274.

[174] Nima Sarshar, P Oscar Boykin, and Vwani P Roychowdhury. "Percolation search in power law networks: Making unstructured peer-to-peer networks scalable". In: *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.* IEEE. 2004, pp. 2–9.

[175] *Scaling Ethereum through Layer2s*. URL: https://ethereum.org/en/developers/docs/scaling#layer-2-scaling. (accessed: 08.02.2024).

[176] Philipp Schindler et al. "Distributed key generation with Ethereum smart contracts". In: *CIW'19: Cryptocurrency Implementers' Workshop*. 2019.

[177] Subhabrata Sen and Jia Wang. "Analyzing peer-to-peer traffic across large networks". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. 2002, pp. 137–150.

[178] Pradip Kumar Sharma, Neeraj Kumar, and Jong Hyuk Park. "Blockchain technology toward green IoT: Opportunities and challenges". In: *IEEE Network* 34.4 (2020), pp. 263–269.

[179] Jie Shen, Ana Lucia Varbanescu, and Henk Sips. "Look before you leap: Using the right hardware resources to accelerate applications". In: *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*. IEEE. 2014, pp. 383–391.

[180] Francisco Jose Couto da Silva et al. "Analysis of blockchain forking on an ethereum network". In: *European Wireless 2019; 25th European Wireless Conference*. VDE. 2019, pp. 1–6.

[181] Jeyasheela Rakkini Simon and K Geetha. "Block Mining reward prediction with Polynomial Regression, Long short-term memory, and Prophet API for Ethereum blockchain miners". In: *ITM Web of Conferences*. Vol. 37. EDP Sciences. 2021, p. 01004.

[182] Mayuran Sivanesan, Anupam Chattopadhyay, and Ronak Bajaj. "Accelerating Hash Computations Through Efficient Instruction-Set Customisation". In: *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*. 2018, pp. 362–367. DOI: 10.1109/VLSID.2018.91.

[183] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. "Exploiting KAD: possible uses and misuses". In: *ACM SIGCOMM Computer Communication Review* 37.5 (2007), pp. 65–70.

[184] Daniel Stutzbach and Reza Rejaie. "Understanding churn in peer-to-peer networks". In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006, pp. 189–202.

[185] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. "Characterizing unstructured overlay topologies in modern P2P file-sharing systems". In: *IEEE/ACM Transactions on Networking* 16.2 (2008), pp. 267–280.

[186] *SubnetDAS proposal*. URL: https://ethresear.ch/t/subnetdas-an-intermediate-das-approach/17169. (accessed: 08.02.2024).

[187] Michael Bedford Taylor. "The evolution of bitcoin hardware". In: *Computer* 50.9 (2017), pp. 58–66.

[188] *The IPFS DHT Reader Privacy Upgrade*. URL: https://discuss.ipfs.tech/t/the-ipfs-dht-reader-privacy-upgrade/16279. (accessed: 08.02.2024).

[189] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. "Blockchain scaling using rollups: A comprehensive survey". In: *IEEE Access* (2022).

[190] Sergei Tikhomirov. "Ethereum: state of knowledge and research perspectives". In: *Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10*. Springer. 2018, pp. 206–221.

[191] Michael Tomasello. *Why we cooperate*. MIT press, 2009.

[192] Michael Tomasello et al. "Two key steps in the evolution of human cooperation: The interdependence hypothesis". In: *Current anthropology* 53.6 (2012), pp. 673–692.

[193] Dennis Trautwein. *Libp2p*. URL: https://github.com/plprobelab/network-measurements/blob/master/results/rfm2-uptime-and-churn.md. (accessed: 08.02.2024).

[194] Dennis Trautwein et al. "Design and evaluation of IPFS: a storage layer for the decentralized web". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 739–752.

[195] Dennis Trautwein et al. "IPFS in the Fast Lane: Accelerating Record Storage with Optimistic Provide". In: *https://gipplab.org/wp-content/papercite-data/pdf/trautwein2024.pdf* ().

[196] Pascal Traverse. "Airbus and atr system architecture and specification". In: *Software diversity in computerized control systems*. Springer, 1988, pp. 95–104.

[197] R Fox Vernon. "A brief history of resilience: From early beginnings to current constructions". In: *Community planning to foster resilience in children*. Springer, 2004, pp. 13–26.

[198] Dimitris Vyzovitis and Yiannis Psaras. "GossipSub: A Secure PubSub Protocol for Unstructured, Decentralised P2P Overlays". In: (2019).

[199] Dimitris Vyzovitis et al. "Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks". In: *arXiv preprint arXiv:2007.02754* (2020).

[200] DC Walden and J McQuillan. "The arpanet design decisions". In: *Computer Networks* 1 (1977).

[201] Luqin Wang and Yong Liu. "Exploring miner evolution in bitcoin network". In: *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16*. Springer. 2015, pp. 290–302.

[202] Ping Wang et al. "Analysis of Peer-to-Peer botnet attacks and defenses". In: *Propagation phenomena in real world networks* (2015), pp. 183–214.

[203] Shenyquan Wang, Dong Xuan, and Wei Zhao. "On resilience of structured peer-to-peer systems". In: *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*. Vol. 7. IEEE. 2003, pp. 3851–3856.

[204] Taotao Wang et al. "Ethna: Analyzing the Underlying Peer-to-Peer Network of the Ethereum Blockchain". In: *arXiv preprint arXiv:2010.01373* (2020).

[205] Wenbo Wang et al. "A survey on consensus mechanisms and mining strategy management in blockchain networks". In: *Ieee Access* 7 (2019), pp. 22328–22370.

[206] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[207] Maximilian Wohrer and Uwe Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity". In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE. 2018, pp. 2–8.

[208] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[209] Raymond Lei Xia and Jogesh K Muppala. "A survey of bittorrent performance". In: *IEEE Communications surveys & tutorials* 12.2 (2010), pp. 140–158.

[210] Xiangbin Xian et al. "Improved consensus mechanisms against censorship attacks". In: *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. IEEE. 2019, pp. 718–723.

[211] Mengyuan Zhang et al. "Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks". In: *IEEE Transactions on Information Forensics and Security* 11.5 (2016), pp. 1071–1086.

[212] Qiheng Zhou et al. "Solutions to scalability of blockchain: A survey". In: *Ieee Access* 8 (2020), pp. 16440–16455.