Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Understanding Peer-Discovery in ETH 2.0

Master's Thesis

Samuel Käser

skaeser@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Lucianna Kiffer, Yann Vonlanthen
Prof. Dr. Roger Wattenhofer

August 10, 2023

Acknowledgements

I am heartily thankful to my supervisors, Dr. Lucianna Kiffer and Yann Vonlanthen, whose encouragement, guidance, and support from the initial to the final stages enabled me to develop an understanding of the subject. Their expertise in the realm of Ethereum and P2P networks proved invaluable in shaping this research work.

I would like to express my gratitude to the DISCO. Their resources and an intellectually stimulating environment have contributed immensely to the successful completion of my thesis. The discussions, brainstorming sessions, and thoughtful critiques provided by the team were truly beneficial in refining the scope and depth of my work.

Lastly, I acknowledge my peers for their unwavering support and for challenging me to continually enhance the quality of my work. Their camaraderie made the journey towards the completion of this master's degree an enriching experience.

Abstract

In recent years, the blockchain domain has witnessed significant advancements, with the Ethereum 2.0 network emerging as a focal point of interest. This research delves deeply into the intricate behaviors exhibited by nodes within the network operating under the discv5 discovery protocol. As Ethereum undergoes a pivotal transition from the traditional Proof-of-Work (PoW) model to the more sustainable Proof-of-Stake (PoS) paradigm, understanding the underlying dynamics of its peer-to-peer (P2P) network becomes increasingly vital.

To capture a comprehensive picture of this environment, this thesis employed a specialized P2P crawler, a significant contribution built by the author. This tool was instrumental in gathering granular data on Beacon nodes, as well as their associated routing tables, painting an unprecedented and comprehensive picture of the networking layer. One of the standout findings from this research is the remarkable adaptability of these nodes. This adaptability is most evident in the nodes' propensity for frequent updates to their Ethereum Node Records (ENRs), suggesting a network in constant flux and evolution.

Furthermore, the research brings to the fore the significance of the Node Discovery Protocol version v5.1. This protocol, as the data suggests, plays a central role in bolstering node discovery and facilitating seamless communication within the Ethereum 2.0 framework.

In conclusion, this research not only underscores the complexities and nuances of the Ethereum 2.0 network but also highlights the imperative for continuous monitoring, research, and adaptation. The author's contributions in developing the P2P crawler and elucidating the networking layer dynamics are pivotal. As Ethereum 2.0 continues to carve its niche in the blockchain ecosystem, understanding and optimizing node behavior will be paramount in ensuring the network's long-term resilience, scalability, and success.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Motivation . . . 1

2 Background 2

2.1 P2P . . . 2

2.2 Proof of Work . . . 3

2.3 Proof of Stake in Ethereum 2.0: The Beacon chain . . . 4

2.4 Ethereum Node Record (ENR) . . . 5

2.5 Networking Stack . . . 7

2.5.1 Discovery domain . . . 7

2.5.2 Gossip domain . . . 9

2.5.3 Req/Resp domain . . . 10

3 Gathering Network Data 12

3.1 Network Crawler . . . 12

3.2 Assessing Node Connectivity . . . 13

4 Results 14

4.1 ENR and Node Monitoring Insights . . . 14

4.2 Analysis of Node and ENR Continuity Across Crawls . . . 14

4.3 ENR analysis . . . 17

4.3.1 eth2 key-value distribution . . . 21

4.4 Routing table analysis . . . 22

4.4.1 Routing table size and reactivity . . . 24

4.5 Ipv6 . . . 27

5 Discussion/Outlook **29**

Bibliography **31**

Chapter 1

Introduction

1.1 Motivation

The motivation for this thesis stems from the need to understand the behavior trends in nodes running discv5, a discovery protocol used in Ethereum 2.0. By building a P2P crawler to gather information about nodes in the Beacon network and their routing tables, we aim to answer several key questions. These include understanding the total count of nodes/IPs, the percentage of nodes that can be reached and confirmed as online, the statistics on the response of FINDNODE requests, and the duration for which "old" information stays in node's routing tables.

Furthermore, we aim to delve into the behavior of high churn ENRs from single IPs, the longevity of node responsiveness to pings, and the overlap of ENR and IPs between routing tables. We also intend to investigate the reasons behind the changes in ENRs associated with the same IP.

In conclusion, the motivation for this thesis lies in the need to understand the dynamics of the P2P network that underpins the Ethereum 2.0 system. By exploring the behavior trends in nodes running discv5, we hope to contribute to the ongoing evolution of blockchain technology and its transition towards more structured and secure network models.

Chapter 2

Background

2.1 P2P

Peer-to-Peer (P2P) networks are one of the fundamental aspects of blockchain technology and more specifically Ethereum. In the context of blockchains, a P2P network refers to a decentralized network of nodes (individual machines), where each node has equal authority and operates in a non-hierarchical way. In contrast to a traditional client-server model, where the server has a superior role and clients interact mainly with the server, in a P2P network, all nodes interact directly with the other nodes without the need of an intermediate authority.

Ethereum and other networks leverage the numerous advantages of P2P networks to maintain copies of the distributed ledger:

Decentralization: Decentralization is the main principle of Ethereum. A P2P network makes sure that no single authority has control over the entire network. Each machine (node) in the network can validate transactions and blocks, allowing for a democratized and transparent system.

Resilience and Redundancy: Because of its distributed nature, a P2P network is highly resilient to failures. If a node goes offline, it does not affect the overall functioning of the network because other nodes maintain copies of the distributed ledger. This redundancy also enhances the security of the network, as manipulating the ledger would require altering more than half of all the copies across the network – with thousands of nodes distributed all over the world, the costs of achieving this far outweigh the potential gains.

Scalability: P2P networks can scale horizontally, which means new nodes can be added to the network without the need for substantial reconfiguration or central planning.

Privacy and Anonymity: While all transactions are transparent and traceable on the distributed ledger, the P2P nature of the network can provide a degree of privacy and anonymity. A user's identity is often represented by their public key, decoupling their real-world identity from their actions on the network.

In the P2P network of a distributed ledger, nodes validate new transactions, propose the next blocks, and confirm the validity of other blocks according to a consensus algorithm (like Proof-of-Work or Proof-of-Stake). This way, the distributed ledger maintains its integrity and immutability, rendering it a robust solution for decentralized applications, cryptocurrencies, smart contracts, and many other uses.

2.2 Proof of Work

Consensus algorithms are used to confirm and validate transactions and therefore play a crucial role in maintaining the integrity and security of the network. Proof of Work (PoW) is a consensus algorithm still used by Bitcoin and previously also by Ethereum. Its primary purpose is to deter cyber-attacks, such as Distributed Denial of Service (DDoS) attacks, which exhaust a system's resources by sending numerous fake requests.

The PoW mechanism operates by requiring network participants (miners) to perform complex computational tasks. These tasks involve finding a nonce (an arbitrary number used only once) that, when hashed with the block content, produces a hash value that meets certain predefined conditions. Typically, the condition is that the hash value is less than a specific target value. The difficulty of finding such a nonce is adjustable and is what determines the computational effort needed.

This process is often termed as solving a mathematical puzzle, but in reality, it's more of a trial-and-error operation, given the one-way nature of cryptographic hash functions. The node that first finds a valid nonce (a solution to the "puzzle") broadcasts the new block and its solution to the network. The other nodes can easily verify the solution by hashing the proposed combination of nonce and block content and checking if it meets the target condition.

Once a valid block is proposed and accepted by the network, the miner is rewarded with a certain amount of cryptocurrency. This incentivizes miners to contribute their computational resources to the network.

The name "Proof of Work" comes from the fact that nodes prove they have expended computational work by solving the required task. This computational work serves the two main purposes of confirming transactions and adding them to the distributed ledger as well as deterring malicious activity by making it computationally and thus financially costly to attack the network. [1]

2.3 Proof of Stake in Ethereum 2.0: The Beacon chain

Launched in 2020, the Beacon chain serves as the prototype for Ethereum's transition from a proof-of-work to a proof-of-stake consensus mechanism. This pioneering proof-of-stake blockchain was designed to confirm the viability and stability of proof-of-stake consensus prior to its integration into the Ethereum Mainnet, thus operating concurrently with the original proof-of-work Ethereum blockchain.

Constructed from "empty" blocks, the Beacon Chain needed to be reconfigured to import transaction data from execution clients. The data is then assembled into blocks, which are subsequently organized into a blockchain using the proof-of-stake consensus protocol. Concurrently, the original Ethereum clients relinquished their mining, block propagation, and consensus functionalities, entrusting these critical tasks to the Beacon Chain. This significant transition is referred to as "The Merge".

Following the Merge, the Ethereum client underwent a significant transformation, now consisting of two distinct layers. This separation becomes especially pivotal in the context of the Merge.

The first is the Consensus-layer, which is responsible for the consensus, specifically block seal validity, and the fork choice rule. With The Merge enabling PoS consensus driven by the Beacon chain, this layer is represented by a modified Beacon chain client. As it stands now, the Beacon chain functions as this consensus layer, forming a peer-to-peer network of consensus clients that manage block gossip and consensus protocols. The Consensus-layer is tasked with maintaining the consensus chain (Beacon chain) and the fork choice rule by processing consensus blocks (Beacon blocks) and attestations received from other peers connected to the attestation sub-network, a part of the beacon chain network. Each Beacon block contains an execution payload, which has a list of transactions and other necessary data for execution and validation. If the payload is invalid, then the Beacon block is also considered invalid. To verify this validity condition, the consensus-layer sends the payload to the execution-layer.

The second is the Execution-layer, responsible for transaction bundling, execution, and state management. This layer is represented by modified pre-merge PoW clients. The original clients now form this execution layer, which handles gossiping and executing transactions, as well as managing the state of Ethereum. The Execution-layer is responsible for assembling an execution block (a former PoW block with stubbed ethash fields), verifying pre-conditions, executing transactions, and verifying post-conditions. After execution, the block is inserted into the execution chain (the execution-layer blockchain, or the pre-merge Ethereum chain) and the post-block state is stored in the execution state storage (the Ethereum state storage as we know it today).

Intercommunication between these two layers is facilitated via the Engine

API. Unlike PoW, where block creation probability is tied to a miner's computational power, PoS determines this probability based on the number of tokens (in Ethereum's case, ETH) a participant holds or "stakes", effectively acting as virtual mining power.

In Ethereum's PoS, to become a validator, a node must stake a certain amount of Ether into the network as collateral. It can do so by transferring a minimum of 32 Ether to the staking smart contract. The staking-process is divided up into slots of 12 seconds, where each slot produces one block and every 32 slots starts a new epoch with a checkpoint to attest to. Checkpoints that reach at least $\frac{2}{3}$ of all the attestations are considered justified. If the next checkpoint of a justified checkpoint becomes justified too, the older checkpoint reaches finality. If the validator engages in slashable offenses such as double-proposing, up to 100% of the stake can be deducted from the validator's account and result in the validator losing his ability to participate in the staking-process. If a staker wants to participate in the staking-process with less than 32 ETH, you can send your ETH to a staking pool and let the pool operate the node. The pool takes a percentage of the accumulated staking rewards for his services.

The benefits of PoS include:

Energy Efficiency: PoS requires significantly less computational power and hence, less energy compared to PoW. This makes the blockchain more sustainable in the long run.

Security: PoS secures the network through the financial commitment of validators. The potential loss of their staked ETH discourages malicious actions.

Scalability: By reducing the need for computational power, PoS potentially allows for a higher transaction throughput, improving the scalability of the network. [2]

2.4 Ethereum Node Record (ENR)

Ethereum Node Records (ENRs) are a fundamental component of Ethereum's networking protocol, providing a mechanism for nodes to share and discover information about each other. As defined in EIP-778 [3], an ENR is a signed key-value record that contains the following components:

- A sequence number: This is incremented whenever the record changes, allowing nodes to keep track of the most recent information.
- The node's public key: This is used for identification and cryptographic operations.
- A signature: This is created using the node's private key and serves to verify the authenticity of the record.

- Additional key-value pairs: These hold arbitrary node metadata and can include a variety of information.

The key-value pairs can include the following keys:

- "id": This specifies the name of the identity scheme, such as "v4".
- "secp256k1": This holds the compressed secp256k1 public key of the node, which is 33 bytes long.
- "ip": This holds the IPv4 address of the node, which is 4 bytes long.
- "tcp": This holds the TCP port of the node, represented as a big-endian integer.
- "udp": This holds the UDP port of the node, represented as a big-endian integer.
- "ip6": This holds the IPv6 address of the node, which is 16 bytes long.
- "tcp6": This holds the IPv6-specific TCP port of the node, represented as a big-endian integer.
- "udp6": This holds the IPv6-specific UDP port of the node, represented as a big-endian integer.
- "eth2": This field encapsulates an ENRForkID object. This object is SSZ encoded and comprises three primary components: fork_digest, next_fork_version, and next_fork_epoch. The fork_digest is a 4-byte value derived from the compute_fork_digest function, which takes into account the node's current fork version and the genesis_validators_root. The genesis_validators_root is a static value found in state.genesis_validators_root.

  The next_fork_version is a 4-byte value representing the version of the upcoming planned hard fork. If there's no impending fork, this value mirrors the current fork version, indicating the absence of any future forks.

  The next_fork_epoch, an 8-byte value, signifies the epoch when the next planned fork will take place. In scenarios where no future fork is on the horizon, this value is set to FAR_FUTURE_EPOCH, signaling the indefinite postponement of any forks.

  For seamless network operations, clients are encouraged to connect with peers whose fork_digest, next_fork_version, and next_fork_epoch match their local values. While clients have the discretion to connect with peers sharing the same fork_digest but differing in next_fork_version or next_fork_epoch, they must be cautious. If the ENRForkID isn't updated to match before the earlier next_fork_epoch of the two clients, their interaction will be disrupted from that epoch onward.

- "attnets": This key is used to discover peers participating in particular attestation gossip subnets. The current initial phase 0 of ETH 2.0 is distinct because it doesn't have shard committees, leaving the attestation subnets, identified as beacon_attestation_subnet_id, without a stable foundation. To counteract this instability, each beacon node is advised to maintain a subscription to a predetermined number of subnets, known as SUBNETS_PER_NODE, for a duration set by EPOCHS_PER_SUBNET_SUBSCRIPTION (default value =256, 27h) epochs.

  The selection of these subnets isn't random but is determined by the node's ID. A specific function named compute_subscribed_subnets(node_id, epoch) has been provided to facilitate this selection. This function uses a prefix of the node's ID and a permutation seed, which is derived from the current epoch combined with the node's unique offset. This combination ensures that each node's subnet selection is both deterministic and shuffled.

All keys except "id" are optional, and a record can still be valid without endpoint information as long as its signature is valid. The ENR is encoded using Recursive Length Prefix (RLP) encoding, and its textual form is the URL-safe base64 encoding of its RLP representation, prefixed by "enr:". [4]

2.5 Networking Stack

In order to be able to run a full node on the Ethereum 2.0 network, it's required to run two different clients concurrently, one for the execution layer and one for the consensus layer. Both clients connect to their own set of peers and participate therefore in two distinct networks. Let's dig deeper into the three domains of a consensus layer client, visualized in figure 4.1.

2.5.1 Discovery domain

The Beacon chain's discovery domain, underpinned by the discv5 protocol, is a sophisticated mechanism for node discovery and communication, drawing inspiration from the Kademlia distributed hash table (DHT) and tailoring it to its specific needs.

Central to discv5 is the Ethereum Node Record (ENR), which holds essential information about each node. For a node to be relayed within the network, it must minimally provide an IP address and a UDP port. The protocol employs a unique method to determine the "distance" between two node IDs, through the logarithmic distance, using a bitwise XOR operation.

The Kademlia protocol's distance metric is employed to identify the node "closest" to a given target node ID. When a node is queried for a target ID, it

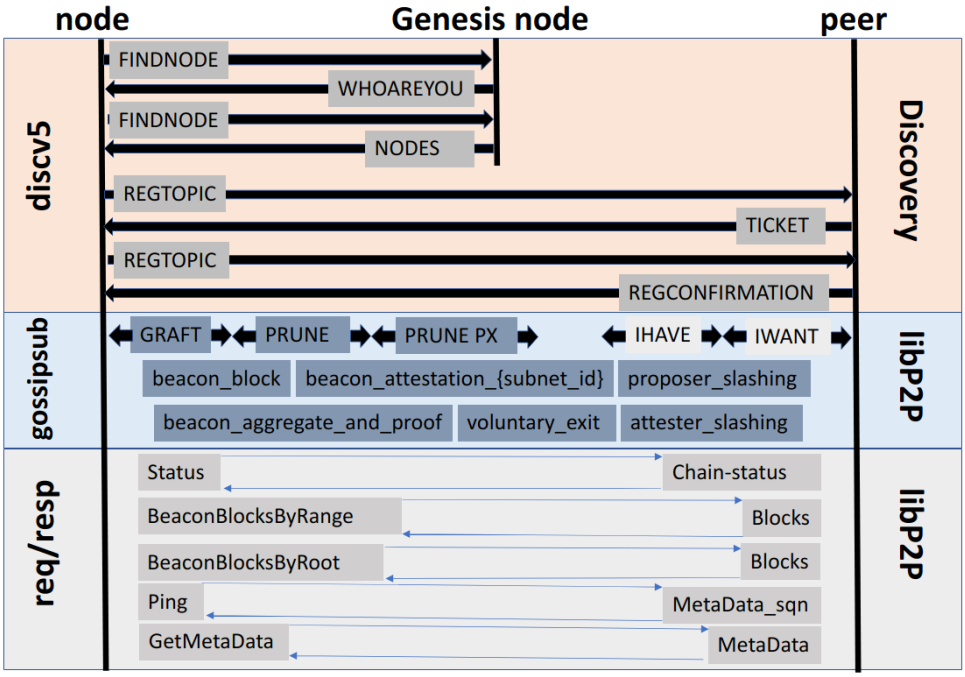

Figure 2.1: The networking stack of a consensus client.

responds with the contact details of the nodes it knows that are closest to the target based on this XOR distance metric. The querying node then continues the lookup process iteratively, querying these closer nodes until the target node is found or no closer nodes are identified.

A significant aspect of the protocol is its messaging system. The FINDNODE message, for instance, is sent by a node (Node A) to another (Node B) to discover nodes close to a target. Node B responds with NODES messages containing the nodes at the queried distance. If the response is insufficient, Node A varies the distance to retrieve more nodes from adjacent k-buckets on Node B. This iterative process, inspired by Kademlia's lookup mechanism, ensures efficient node discovery.

Kademlia's influence is further evident in the organization of data within the DHT. Nodes maintain "k-buckets", array-like data structures that store contact information for other nodes. The number of k-buckets a node manages is determined by the bit length of node IDs. Beacon nodes have a 256-bit node ID, resulting in 256 k-buckets. The position of a node within the k-bucket array is determined by the bit difference between its ID and the managing node's ID. The index of the k-bucket in which a node resides corresponds to the position of the leftmost differing bit between the two IDs. Each k-bucket can hold a maximum of "k" entries.

Beyond these functionalities, discv5 places a strong emphasis on security through its handshake and encryption mechanisms. Discovery communication is encrypted and authenticated using session keys, established in a handshake process. This handshake can be initiated by either side of communication at any time. The handshake process involves several steps, starting with a node sending a message packet. If the receiving node doesn't recognize the session keys, it responds with a WHOAREYOU packet, challenging the initiator. The initiator then sends a handshake message packet containing its identity proof and other essential details. Once the handshake is successful, both nodes establish new session keys for encrypted communication. This ensures that only nodes that have successfully completed the handshake can communicate, providing a layer of security against potential adversaries.

In essence, the Beacon chain's discovery domain, with its discv5 protocol, offers a robust mechanism for node discovery, communication, and security. It seamlessly integrates the principles of the Kademlia DHT while adapting it to its unique requirements and adding layers of encryption and authentication for enhanced security. [5, 4, 6]

2.5.2 Gossip domain

[7] The gossip-domain, which is part of the libp2p framework, consists of the gossipsub-protocol. This domain is responsible for the distribution of various types of messages across the network. Each node is connected to multiple peers for different topics it's subscribed to.

There are six different topics that nodes can subscribe to: beacon_block, beacon_aggregate_and_proof, voluntary_exit, proposer_slashing, attester_slashing and beacon_attestation_{subnet_id}. Each of these topics serves a specific purpose in the network.

The beacon_block topic is used to distribute new signed beacon blocks, and every node is subscribed to that topic. There are a bunch of requirements the block must pass in order for the node to propagate the message to its peers.

To distribute aggregated attestations to subscribed nodes, such as validators, the beacon_aggregate_and_proof topic is used. The voluntary_exit topic is used to propagate messages about signed voluntary validator exits. To propagate proposer or attester slashings, nodes use the proposer_slashing or attester_slashing topics.

Nodes can also propagate unaggregated attestations to their peers in a specific subnet before they are being aggregated, by using the beacon_attestation_{subnet_id} topic. This topic is used to share attestations only among the subnet-members such that attestation-aggregation only happens on the subnet level. The subnet-membership is defined in the attnets entry of the ENR.

Once the node has peers with valid session keys, it can initiate full connections in the gossip-domain by sending GRAFT messages to them. This means that the node adds them to the mesh for a specific topic and will forward any valid messages of that topic to them. If the node exceeds its target range for connections, it disconnects from older peers by sending them a PRUNE message, which informs the peer that the full connection has been disbanded. To help the disconnected peer find new peers, the PRUNE PX message allows the node to append a list of node IDs as alternatives that the receiver can connect to.

The IHAVE and IWANT messages are the gossip that our node can send to any peer that it is aware of, independent from whether they are in one of the topic meshes or not. IHAVE lets other nodes know about messages that the node has received and could be shared upon the IWANT request.

Overall, the gossip domain is used to propagate information that has to be spread quickly over the entire network. It plays a crucial role in maintaining the health and efficiency of the network.

2.5.3 Req/Resp domain

The Request/Response (Req/Resp) domain is a key component of peer-to-peer interactions in a Beacon chain client, facilitating specific information exchanges such as Beacon blocks associated with distinct root hashes. This domain is based on the principles of the libp2p framework, which outlines a detailed structure for constructing a Req/Resp protocol.

A fundamental operation in this domain is the 'status' handshake request, initiated by the dialing client. This operation includes the exchange of the current version of the Beacon chain, comprising several components: fork_digest, finalized_root, finalized_epoch, head_root, and head_slot. This mutual sharing of status information serves as the foundation for further interactions. At the beginning of every full connection, the nodes have to synchronize first, by sending a status message where they inform each other about their current version of the chain called fork_digest. In case the two chains are irreparably disjoint, the nodes won't proceed with the connection process.

In scenarios where there is an agreement between the nodes' states or one node is slightly lagging, the less updated node can send a BeaconBlocksByRange request. This request aims to retrieve all the blocks that the more updated node possesses, thereby aligning the nodes' states.

Another notable request message is BeaconBlocksByRoot, enabling requesters to ask for specific blocks. Here, the requester sends a list of missing block roots, aiding in precise and efficient information retrieval.

Further enhancing the robustness of the Req/Resp domain is the 'ping' protocol, which allows a node to check the responsiveness of a peer by sending their

MetaData.seq_number. If a response is not received, the requesting node disconnects from the peer, ensuring the overall network's reliability. Furthermore, if the MetaData.seq_number implies that the local record of the peer's metadata is outdated, the requester can use the GetMetaData message to fetch the most recent metadata from its peer.

This dynamic interaction mechanism of the Req/Resp domain ensures a consistently updated and efficient beacon chain. It enables nodes to maintain a coherent state and facilitates swift recovery in case of any discrepancies, thus significantly contributing to the overall performance and reliability of the Ethereum 2.0 network.

Chapter 3

Gathering Network Data

To delve deeper into the discovery layer of the Beacon chain network's intricacies, our objective was to compile a comprehensive dataset comprising Ethereum Node Records (ENRs) and the routing tables of all accessible nodes. Our approach involved the creation of a crawler capable of accessing the routing tables of every reachable node within a span of 70 minutes. This crawler was then scheduled to operate weekly to identify new nodes. Additionally, to gather connectivity metrics, we initiated a ping to all identified ENRs on an hourly basis.

3.1 Network Crawler

Initiating its operations, the crawler establishes a connection with a predetermined bootnode, subsequently accessing its routing table. As delineated in the preceding chapter, nodes maintain the ENRs of their associated peers across 256 distinct buckets. The placement of a peer's Node Id within these buckets is determined by its XOR distance, with greater distances corresponding to higher bucket numbers. Notably, the initial 240 buckets for most nodes remain unoccupied. This characteristic enabled us to focus our queries on the final 16 buckets to obtain a complete routing table for a node. Upon successfully connecting with a node via the discv5 protocol, we dispatch a FINDNODE message for each bucket. It's worth noting that nodes provide the content of only one bucket in response to each FINDNODE message. Consequently, we can retrieve a maximum of 16 unique ENRs from the responding node. Post-crawl, we possess the routing table of every responsive node. This implies that the utilized ENR for addressing the node was current, the node wasn't obstructed by a NAT, and it acknowledged the FINDNODE message. Collating data from these routing tables, we generate a list of unique ENRs, termed as "discovered nodes." Our crawler was configured to operate weekly, hosted on an AWS EC2 instance situated in the us-east-1 region. Presented below is the crawler's pseudocode, where the active tasks list represents a queue of newly identified ENRs awaiting routing table queries:

Create log file and directory for routing tables

Initialize boot node from hardcoded ENR
Initialize data structures for storing nodes and listening ports
Loop:
  If number of active tasks is less than limit
    If there is a node to be processed
      Spawn a new thread to get routing table for the node
      Send node and its routing table to another thread via channel
      Save routing table to a file and update index file
      Add new nodes from routing table to queue of nodes to be processed
    End If
  End If
End Loop if no new nodes have been encountered in the last 60 seconds

3.2 Assessing Node Connectivity

Post-crawl, the nodes identified are subsequently monitored by our connectivity verification system. On an hourly basis, each ENR undergoes a ping via the discv5 protocol. If a connection is successfully established during this process, the ENR is deemed active. This methodology allowed us to produce hourly connectivity metrics for every ENR. Notably, with each new crawl yielding a fresh set of ENRs, the previously monitored set is discarded in favor of tracking the newly identified ENRs.

Chapter 4

Results

4.1 ENR and Node Monitoring Insights

The hourly connectivity data facilitates the creation of the graph depicted in figure 4.1. The chart's orange trajectory represents the count of ENRs that are responsive to PING requests, termed as "active ENRs". A noticeable trend is the immediate decline in active ENRs following each crawl. This decline can be attributed to nodes either becoming inactive or undergoing ENR modifications. Various factors can prompt nodes to alter their ENRs, a topic we'll delve deeper into subsequently. A recurring observation is the stabilization of active ENRs to a consistent range (approximately 16'000 to 17'000) post each crawl.

The connectivity checker yields an hourly dataset of active ENRs. By selecting a specific set, such as the active ENRs from hour 100 depicted in the chart, we can monitor its activity throughout the entire timeline. It's noteworthy that some ENRs active during hour 100 had been active in prior crawls, explaining why the blue trajectory doesn't originate from the y-axis's zero point. The pronounced surge and subsequent decline during the recent crawls indicate that the majority of ENRs active during hour 100 were identified in the latest crawl and were not detected in the subsequent one, leading to their removal from the monitored ENR list. Post this sharp decline, the rate of activity reduction is more gradual, suggesting a subset of nodes that infrequently modify their ENRs compared to the broader node population.

4.2 Analysis of Node and ENR Continuity Across Crawls

To gain a comprehensive understanding of the Ethereum network's dynamics, we expanded our analysis beyond merely tracking active ENRs. By plotting the total number of ENRs encountered in each crawl and subsequently monitoring them in all following crawls, we can discern patterns of persistence and overlap.

In figure 4.2, the blue lines represent the total number of initially encountered ENRs in each crawl. While this number consistently hovers around 52'000 to

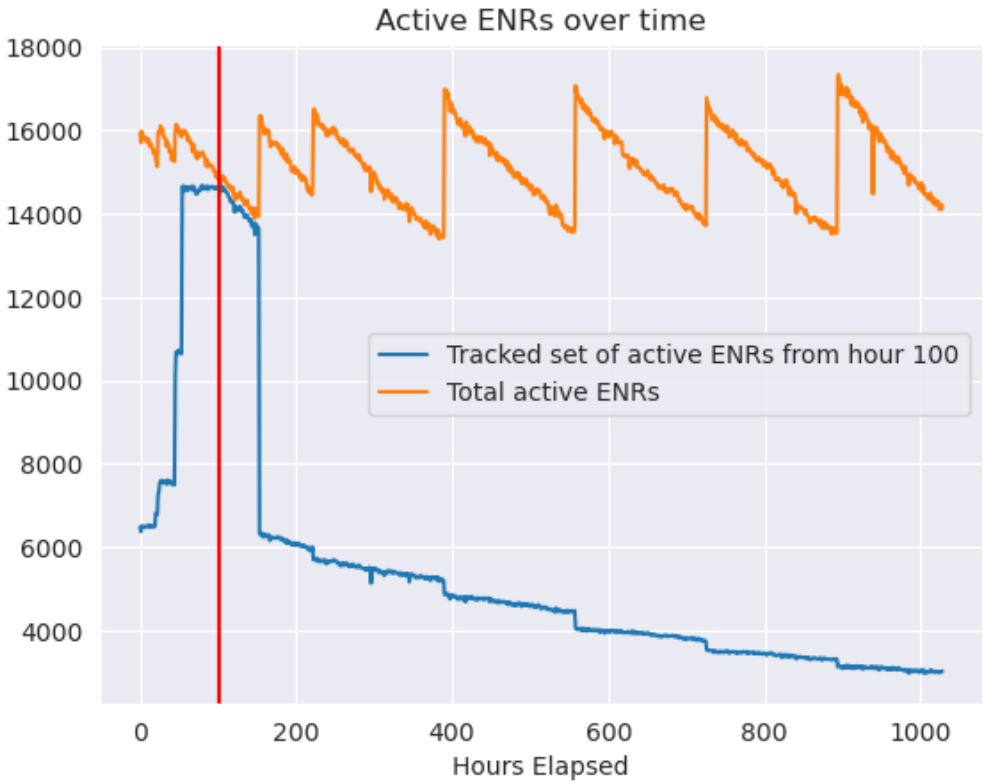

Figure 4.1: The orange line tracks the overall active ENRs that are being updated with every crawl and the blue lines tracks the specific set of ENRs that were active in hour 100, indicated by the red vertical line.

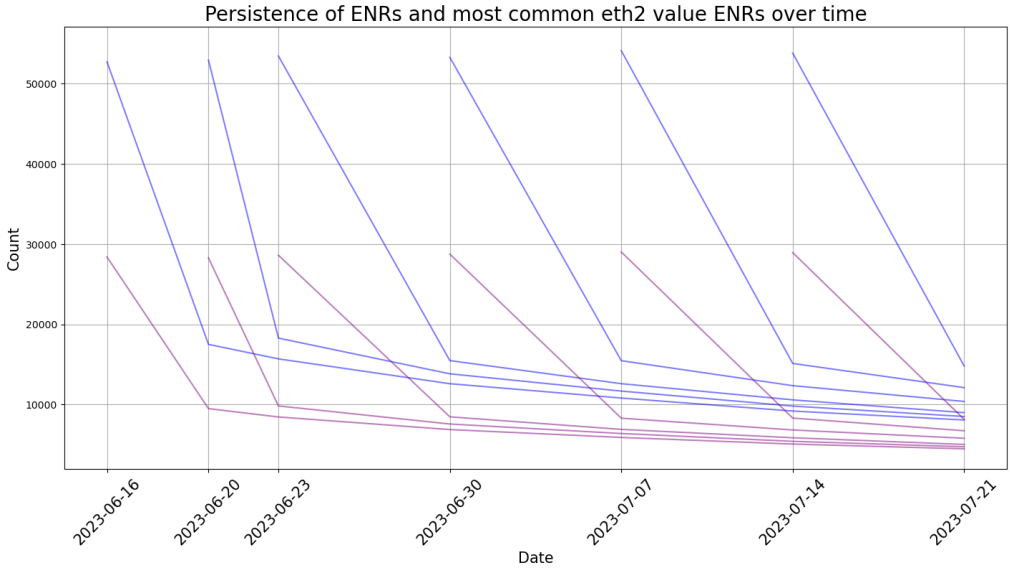

Figure 4.2: The blue lines show the persistence of all ENRs starting from each crawl whereas the purple lines only consider the ENRs stemming from nodes on the Beacon mainnet and running the latest fork version Capella.

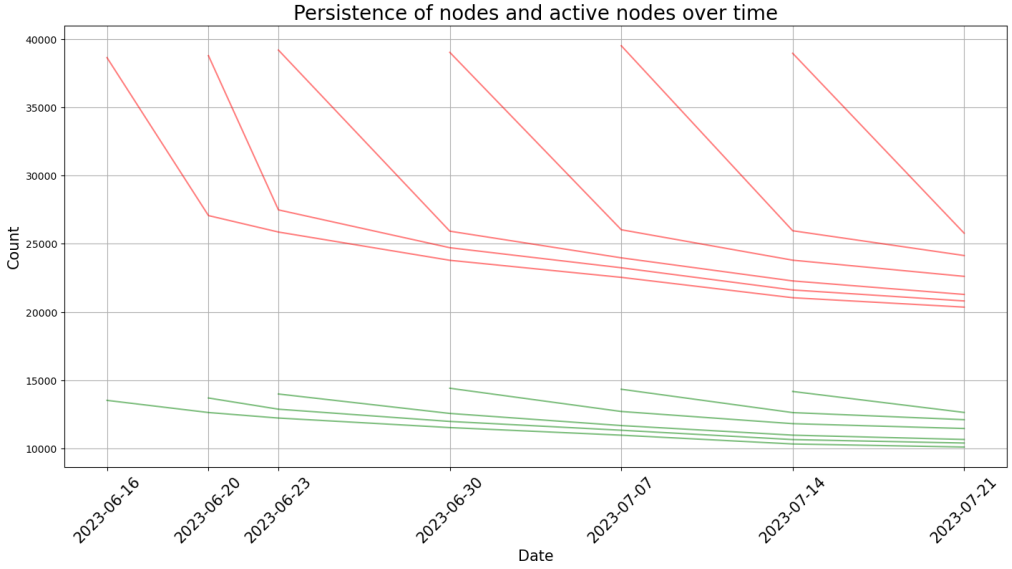

Figure 4.3: The red lines show the persistence of all nodes starting in each crawl whereas the green lines only consider nodes where at least one of its ENRs was registered active during the post-crawl activity-tracking period.

53'000 unique ENRs, a significant drop to a range of 15'000 to 20'000 is observed, eventually stabilizing around 6'000 in subsequent crawls. However, when we filter these ENRs to only include those from nodes running the latest fork version on the beacon mainnet(represented by the purple line, which accounts for approximately 50% of all ENRs in each crawl), a similar relative decline is observed, suggesting a consistent trend.

Decoding the ENRs to extract IP/Port information (from keys such as "ip", "tcp", "udp", "ip6", "tcp6", and "udp6") allows us to group ENRs by unique IP/Port combinations, which we define as nodes. Figure 4.3 showcases the number of these unique combinations (red lines) across crawls. Initially, each crawl reveals about 38'000 to 39'000 unique combinations from the 52'000 to 53'000 ENRs. This number reduces to approximately 26'000 in the subsequent crawl and seems to stabilize around 20'000 in later crawls. To identify active nodes, we consider nodes with at least one ENR deemed active during the period between crawls. The green lines in the figure represent these active nodes, which display greater persistence across crawls compared to the overall node population. The difference between the number of active nodes identified in the initial and final crawls is a mere 3'500, from an initial count of around 13'500.

A key observation is the higher churn rate in ENRs compared to nodes. Nodes "vanish" only when they go offline or modify their IP/Port. In contrast, an ENR can "disappear" due to these reasons or when a node alters other features, such as its public key. This distinction is further highlighted in figure 4.4, which plots the overlap of nodes and ENRs between consecutive crawls. Here, overlap is defined as the intersection of two crawls divided by their union. Notably, node overlap is more than double that of ENRs. The marginally elevated values observed in the initial two crawls can be attributed to the shorter intervals between them.

4.3 ENR analysis

By grouping all identified ENRs based on their IP/Port attributes and decoding the values for every key encountered across these ENRs, we can undertake a comparative analysis of these values within each group. It's noteworthy that several keys are present in only a minority of ENRs, implying that most ENRs lack values for these specific keys. Such infrequently occurring features include photon, diff, galaxy, snap, ncogearthchain, exp, opera, bsc, les, metatechchain, vns, and skyhigh. As a result, the majority of groups exhibit uniformity concerning these features.

In figure 4.5, the distribution among these groups is bifurcated into two distinct categories. The blue bars denote the count of IP/Port groups that appear in a minimum of two different crawls, where all values for a particular feature remain consistent within the group. This consistency indicates that the feature

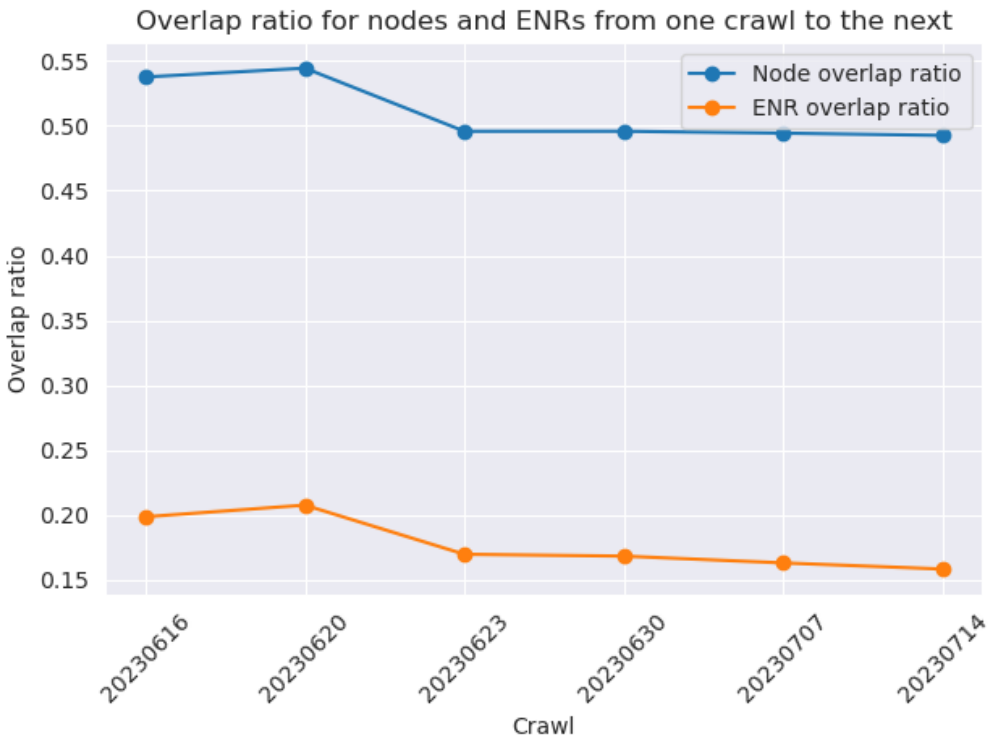

Figure 4.4: The overlap ratio (intersection divided by union) of crawled ENR sets for ENRs and nodes compared with the next crawl.

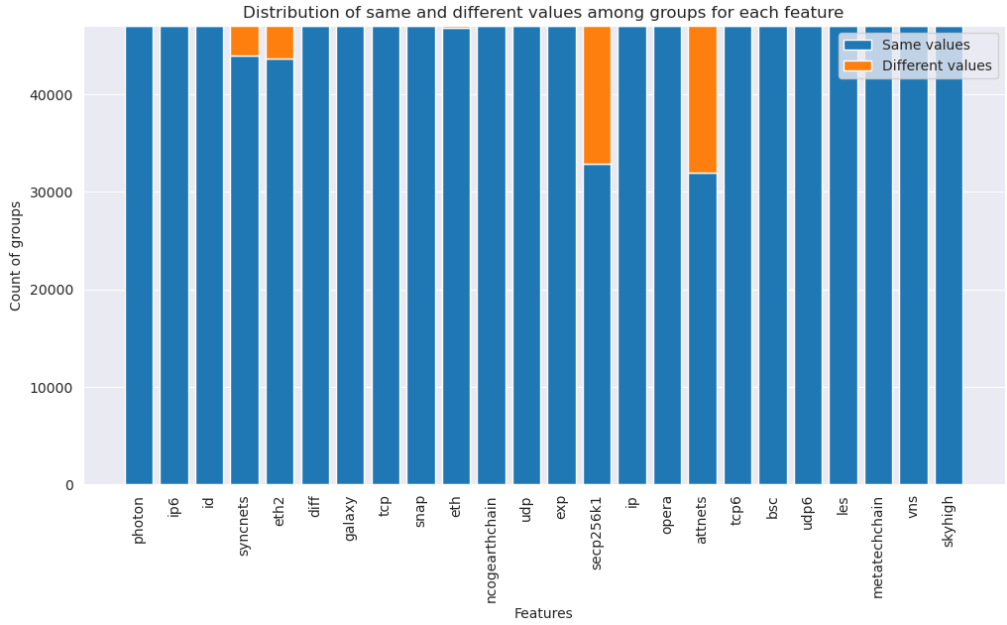

Figure 4.5: Share of nodes that have the same/different value among all ENRs within their group, shown for each feature in the ENR.

remained unchanged over the observed timeframe. Conversely, the orange bars signify the groups where varying values for a specific feature were observed, even though the IP/Port remained consistent. This suggests that while the IP/Port was stable, the feature in question underwent changes.

A notable observation is the alteration in nodes' public keys (secp256k1) and the bitvector that signifies the subnets to which they are subscribed (attnets). This aligns with the consensus specifications [4], which recommend regular modifications to the subscribed nets. It's plausible that many nodes either subscribe to all available nets or refrain from altering their subscriptions within the monitored period, resulting in an unchanged attnets value. Additionally, certain nodes appear to transition between networks (e.g., from a testnet to the mainnet) or modify their current or impending fork versions. This is evident from the groups that display varied values for the eth2 key. As for the significance of other features, our research did not yield substantial insights. Neither the EIP-778 [4] nor the Beacon chain consensus specifications provide explicit mentions of these features.

bba4da96,03000000,0.505723 628941ef,03001020,0.147762 189a8e29,40000040,0.039496 4a26c58b,02000000,0.037019 47eb72b3,90000072,0.025358 824be431,02000064,0.018416 c2ce3aa8,02001020,0.014226 4a26c58b,03000000,0.014069 824be431,03000064,0.010858 bba4da96,04000000,0.007578

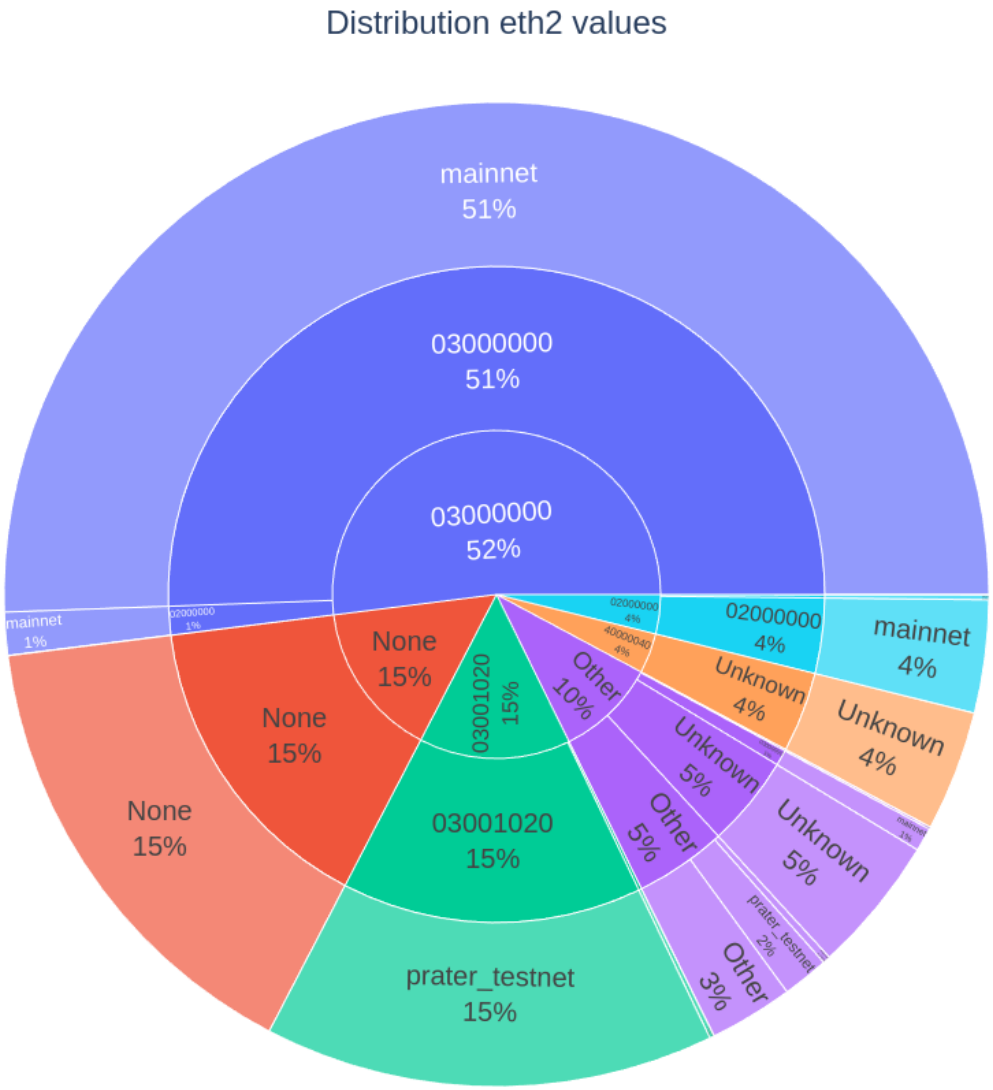

Figure 4.6: Distribution for the decoded values of the eth2 feature from all encountered nodes. The outer ring represents the net, the middle ring represents the current fork version and the inner ring the future fork version.

fork_digest	genesis_validators_root	current_fork_version
0xbba4da96	0x4b363db94e... (mainnet)	03000000 (Capella)
0x628941ef	0x043db0d9a8... (prater_testnet)	03001020 (Capella)
0x189a8e29	Unknown	Unknown
0x4a26c58b	0x4b363db94e... (mainnet)	02000000 (Bellatrix)
0x47eb72b3	0xd8ea171f3c... (bepolia_testnet)	90000072 (Unknown)
0x824be431	Unknown	Unknown
0xc2ce3aa8	0x043db0d9a8... (prater_testnet)	02001020 (Bellatrix)
0x36ba57db	Unknown	Unknown
0x3cfa3bac	Unknown	Unknown
afcaaba0	Unknown	Unknown

Table 4.1: Above are the 10 most common fork_digests, some of which we were able to map to their genesis_validators_root and current_fork_version

4.3.1 eth2 key-value distribution

The decoded value of the eth2 key, as highlighted in the Introduction, can be divided into three distinct sections fork_digest, next_fork_version and next_fork_epoch. The computation of the fork_digest involves the genesis validators root and the prevailing fork version. It's noteworthy that the genesis validators root is distinct for each network. In the context of Ethereum 2.0, there exists a primary mainnet and two actively maintained Beacon chain testnets by client developers, named Prater and Bepolia. The genesis validators roots for these networks are well-documented [8, 9, 10, 11]. In order to be able to map eth2 key-values to the three known nets as well as the current fork version that they are running, we took in a first step the three known genesis validators roots and all possible fork versions as inputs to the compute_fork_digest function. Every possible combination of those inputs resulted in a different fork_digest value that we were then able to compare with the fork_digest values we encountered in the ENRs and map subsequently map the input values to the ENRs.

A visualization of all nodes is presented in figure 4.6. A significant majority of nodes are affiliated with the mainnet and operate on the most recent fork version, Capella:0300XXXX, as cited in [12]. Interestingly, 15% of nodes lack an eth2-key, while an identical percentage, 15%, are associated with the Prater testnet and also utilize the latest fork version. A smaller fraction, 4%, are linked to the mainnet but are yet to transition to the latest fork version, continuing to operate

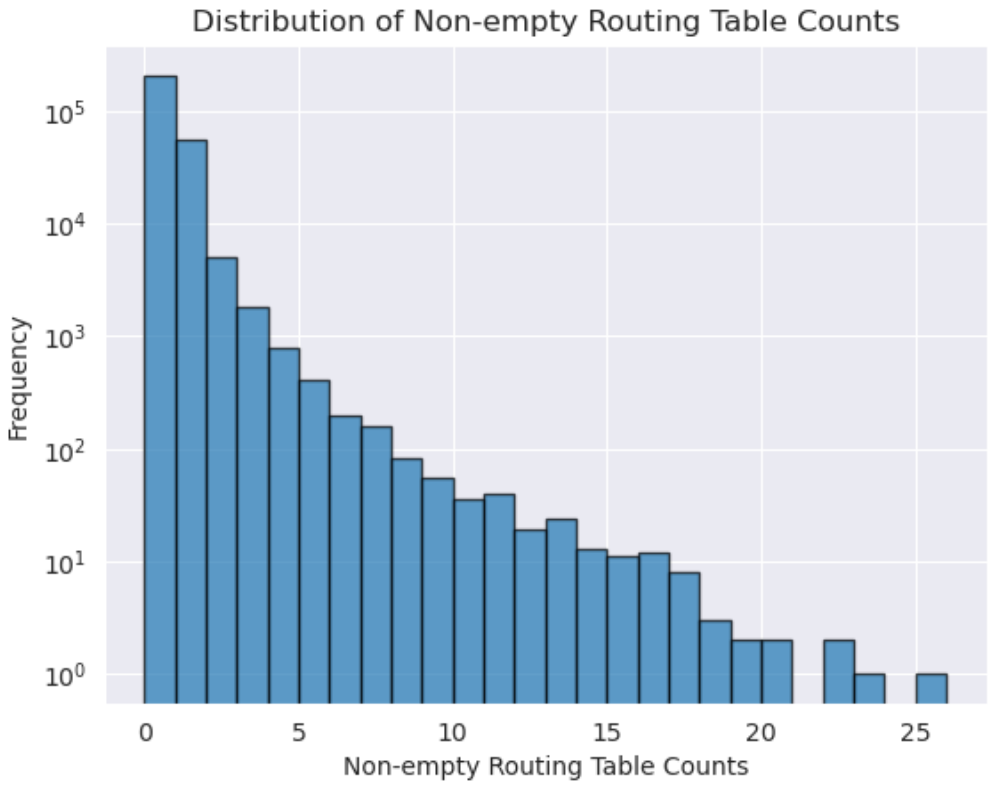

Figure 4.7: Distribution over the amount of ENRs per node that responded to a FINDNODE message, grouped per crawl, from all 7 crawls.

on Bellatrix (0200XXXX) as per [12]. Additionally, 1% of nodes, while currently on the mainnet with Bellatrix, have intentions to transition to Capella in the foreseeable future. To ensure clarity in the visualization, infrequent values were grouped under the "Other" label, which also includes a few nodes from Bepolia. The label "Unknown" signifies our inability to correlate the fork_digest with the genesis validators roots from any of the three recognized networks. In instances where a node displayed multiple ENRs with varying values, the most recent ENR was chosen as the basis for node categorization.

4.4 Routing table analysis

During its operation, the crawler attempts to retrieve a routing table from each ENR it comes across. Across the seven crawls conducted, the number of ENRs encountered ranged from 52'692 to 54'565. A significant portion of these ENRs are inaccessible, and even when an ENR is accessible (i.e., responds to a PING), it doesn't guarantee a response to a FINDNODE request. As a result, the count

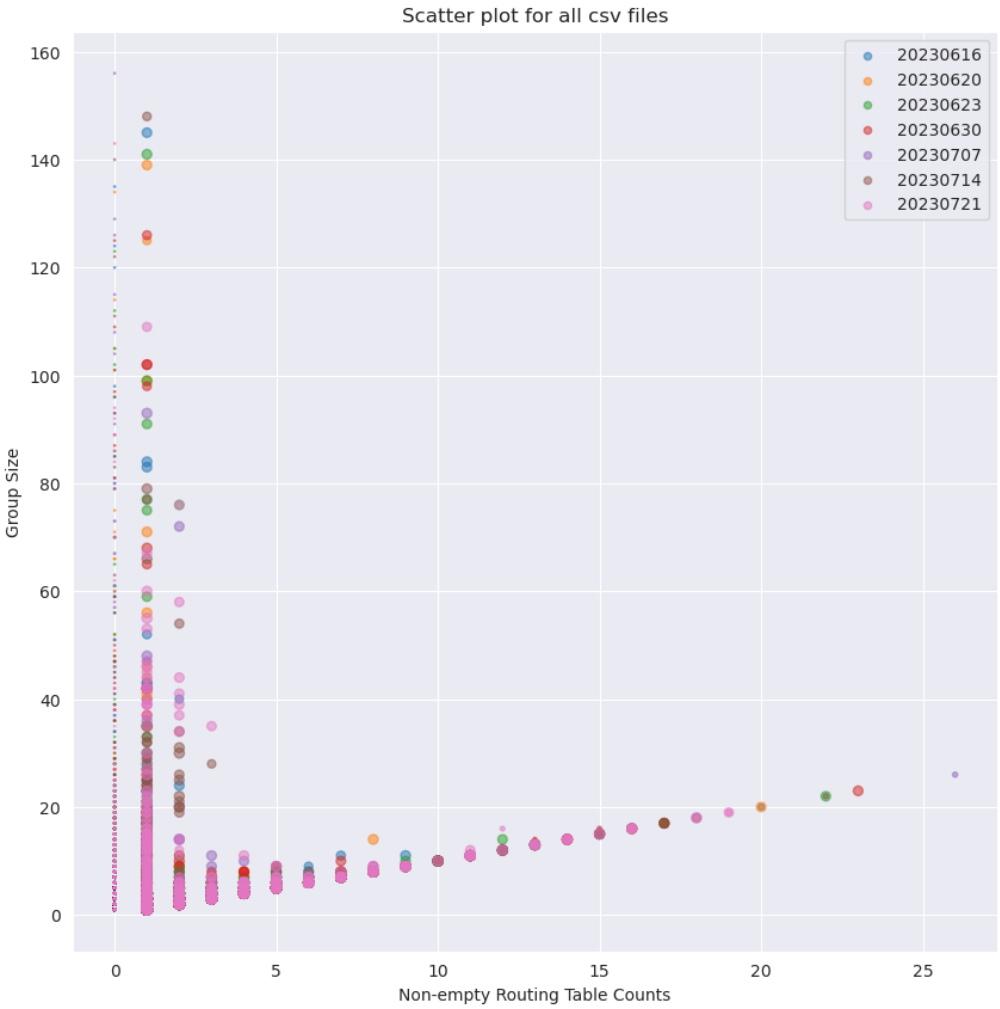

Figure 4.8: The group size on the y-axis refers to the amount of ENRs per node per crawl and on the x-axis is the amount of non-empty routing tables from each node, indicating the responsiveness to a FINDNODE message. The size of the dots relates to the average size of the non-empty routing tables per node. The colors indicate the date of the crawl.

of non-empty routing tables acquired during each crawl fluctuates between 11'549 and 12'143.

Figure 4.7 presents, on a logarithmic scale, the distribution of nodes based on the number of their ENRs with non-empty routing tables. This visualization reveals that a vast majority of nodes, totaling around 205'000 or averaging 30,000 per crawl, lack any ENR with a queryable routing table. Thus, we possess no data regarding their routing tables. Conversely, there are close to 55'000 nodes (averaging about 8'000 per crawl) where only one of their ENRs responded to a FINDNODE message. A few hundred nodes per crawl provided multiple non-empty routing tables.

By analyzing the group sizes, which refers to the number of ENRs captured per node, and juxtaposing this with the count of non-empty routing tables for each node, we derive figure 4.8. In this figure, each dot symbolizes a node. The dot's size represents the mean size of the non-empty routing tables within that group. The most prominent category is evident on the line x=0, which encompasses groups devoid of non-empty routing tables. For these groups, the dot size is set to a standard value of 1. Two nodes with exceptionally high ENR turnover were excluded for clarity. Specifically, one node exhibited between 882 and 913 ENRs per crawl, which would have stretched the visualization. The line x=1 showcases the next prevalent category, consisting of nodes with a single FINDNODE-responsive ENR. For these primary categories, there doesn't appear to be a direct correlation between group size and FINDNODE responsiveness. However, as we progress down the x-axis, a linear correlation emerges for certain nodes, suggesting that some nodes consistently respond with every ENR in their group.

4.4.1 Routing table size and reactivity

Figure 4.9 provides a visual representation of the distribution of sizes for all non-empty routing tables. A significant portion of these tables contain approximately 180 nodes. However, there are some with only a handful of entries, potentially indicating newer nodes that have recently joined the network. Given that nodes can accommodate a maximum of 16 ENRs per bucket, they need to consistently manage these buckets. This involves routinely pinging the ENRs and removing the less responsive ones to accommodate newly identified nodes. To gauge the frequency of updates to these routing tables, we can assess the commonality of a node's non-empty routing tables across different crawls. For each initial crawl, we only took into account nodes that were present in every subsequent crawl after the designated starting crawl.

In figure 4.10, we've illustrated the average commonality across all the nodes under consideration. This overlap is defined by the ratio of the intersection to the union of two routing tables, between the initial crawl and every subsequent

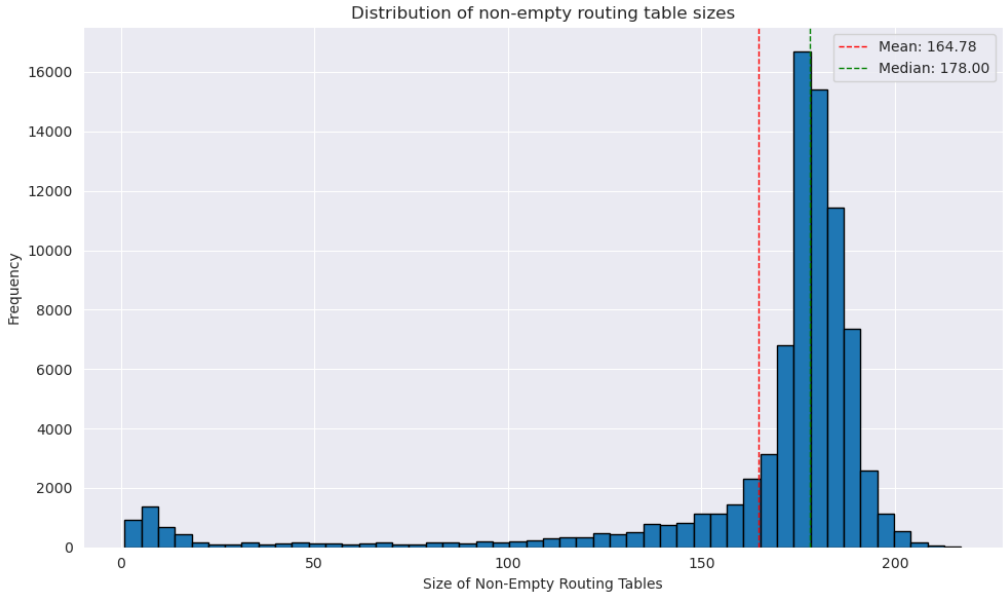

Figure 4.9: Distribution over the sizes of all encountered routing tables with mean and median.

crawl. In cases where a node had multiple routing tables within the same crawl, one was chosen at random. Given that the intervals between the first three crawls are shorter (three to four days), the average overlap from the initial two crawls relative to their subsequent crawls is slightly elevated (around 20%) compared to the overlaps from the remaining starting crawls (ranging from 10% to 15%), which are spaced a week apart. This suggests that nodes are highly reactive, adapting swiftly to an environment where many nodes frequently modify their ENRs.

This heightened reactivity is further evident in figure 4.11. Here, we track the ENRs from the top five nodes with the highest ENR turnover from the first crawl across the routing tables in the following crawls. The node with the highest ENR turnover (located in a datacenter in Dallas, TX) appears in over 700 routing tables (with more than 800 distinct ENRs, implying some nodes have the same node listed under multiple ENRs in their routing tables). The ENR turnover for the other four nodes is considerably lower, with each appearing in 100 to 160 distinct routing tables during the first crawl. The majority of nodes purged the older ENRs from the four smaller groups within the initial four crawls. In contrast, the larger group demonstrated slightly more persistence, still appearing in 200 routing tables during the second crawl but dropping below 100 by the final crawl.

Rather than monitoring ENRs from high-turnover groups that remain active under varying ENRs, we can focus on ENRs from nodes that have been deac-

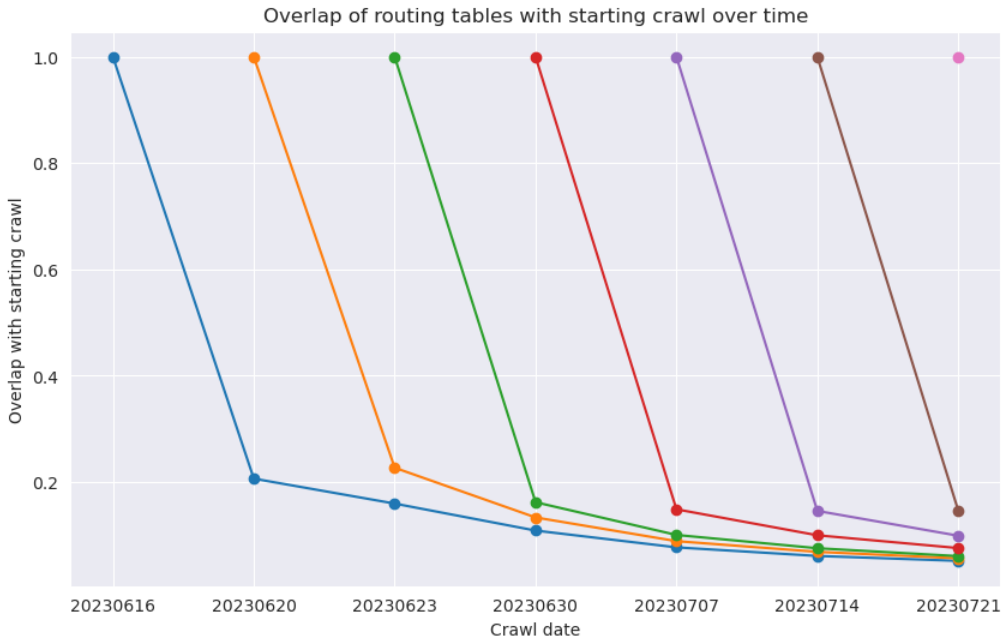

Figure 4.10: Average overlap of routing tables from one crawl compared with subsequent crawls, for the same node.

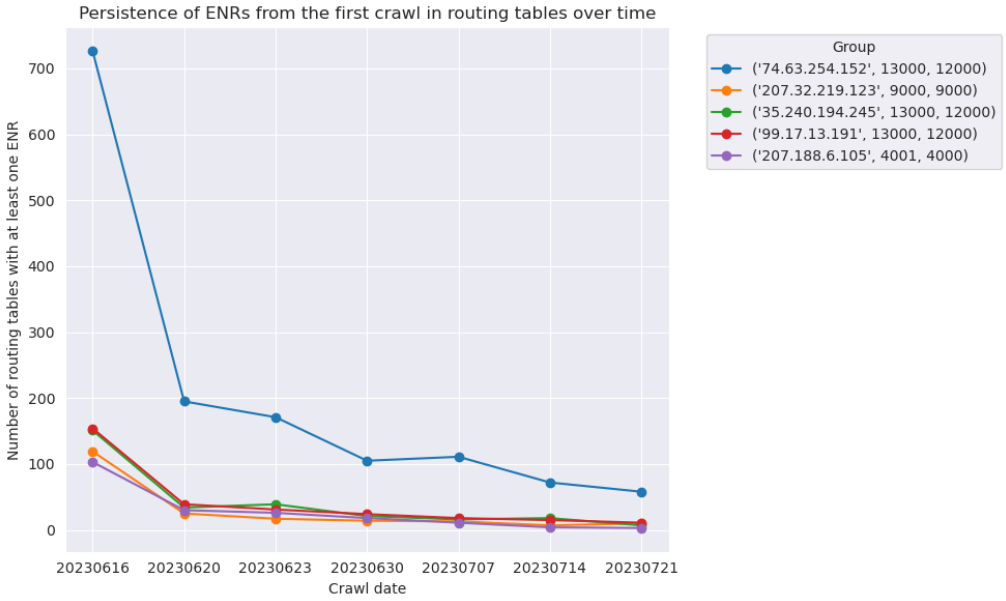

Figure 4.11: Persistence in routing tables from subsequent crawls of ENRs from nodes with a high ENR churn.

tivated. During the activity-tracking phase following a crawl, if any ENR of a node responds to a PING, we deem that node active for that duration. For a node to be classified as deactivated, none of its ENRs should respond to a PING in any subsequent periods. After each crawl, we compile a list of nodes meeting this criterion. From each crawl's list, we then select the three largest groups and monitor the number of routing tables in which these ENRs continue to appear. The findings are illustrated in figure 4.12. While some nodes vanish entirely from all routing tables post-deactivation, others exhibit a degree of persistence. In the legend, the number following the IP, TCP-port, and UDP-port denotes the group size during the crawl when the node was last detected as active.

4.5 Ipv6

The crawler and subsequent activity checks were conducted over IPv4. However, as highlighted in the Introduction, nodes can also store IPv6 connection details within their ENRs. From the entire dataset, we identified 36 unique ENRs that include IPv6 IP/Port data. When these ENRs are grouped based on their IPv6/Port, we discern 10 distinct IPv6 addresses. Some of these addresses are associated with data centers, while others appear to be linked to residential IPs.

Interestingly, with a single exception, each IPv6 address corresponds to just one IPv4 address. This implies that every ENR with a specific IPv6 address X consistently has the same IPv4 address Y. Delving deeper, we found two IPv6 addresses located in China, one in Beijing and another in Shanghai. The Shanghai-based IPv6 address is present in three ENRs. While two of these ENRs share the same IPv4 address, the third one differs. However, all three IPv4 addresses are geographically aligned with the location of the IPv6 address in Shanghai. Notably, the ENR with the Beijing-based IPv6 address has its IPv4 counterpart located in Shanghai, matching the location of the aforementioned Shanghai IPv6.

There's a mix when it comes to the geographical alignment of IPv6-IPv4 pairs. Some pairs share the same location, while others have differing IPv4 and IPv6 locations. When probing the routing tables of these ENRs via either IPv4 or IPv6, the disparities align with those observed when querying distinct ENRs from an identical node consecutively over IPv4. Typically, differences span up to three ENRs in routing tables comprising 180 nodes. This likely stems from the frequent alterations observed in these routing tables.

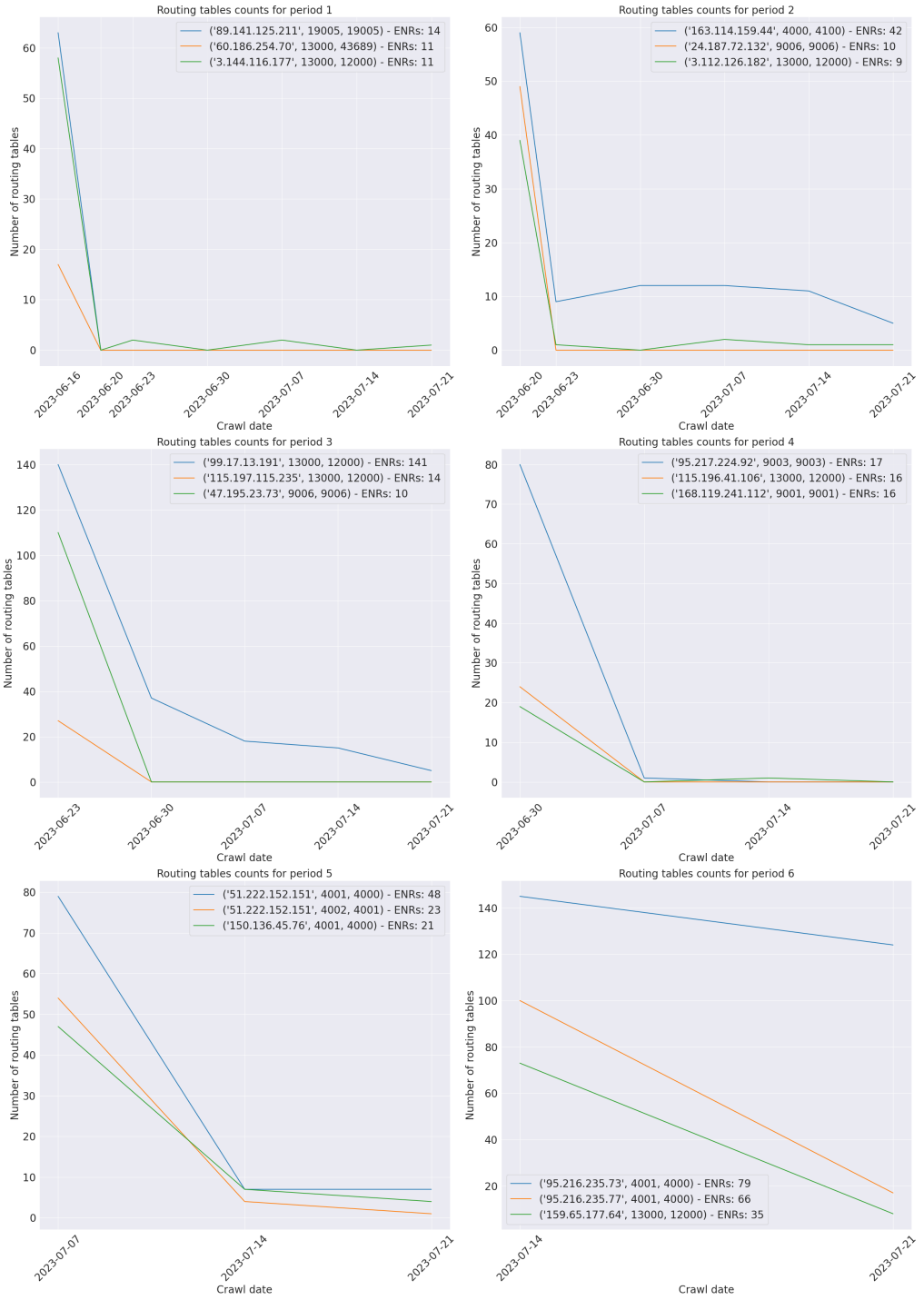

Figure 4.12: Longevity of nodes in routing tables after they get shut down.

Chapter 5

Discussion/Outlook

Upon examining the activity data, we're confronted with a puzzling observation: why is there a significant decline in the number of active nodes from a specific hour by the time of the next crawl? Post each crawl, there's a consistent decline in active ENRs. While one explanation is the deactivation of nodes, leading them to become unresponsive to PINGs, this decline is gradual. If we had continuously tracked all the ENRs from a specific hour without excluding them based on their absence in the subsequent crawl, we'd likely observe this steady decline persisting. Yet, many ENRs, which would have still responded to PINGs, are absent in the routing tables of the next crawl.

So, if nodes verify the vitality of ENRs in their tables through PING messages, why do many ENRs, which would have responded with a PONG, vanish from the routing tables? The answer is embedded in a key aspect of the discv5 protocol [5]. The protocol dictates that if a PONG response from a PINGed ENR suggests a change in the ENR, the node performing the liveness check should retrieve the new record and update its routing table. This mechanism explains the absence of seemingly active ENRs from node routing tables.

Another intriguing point is the disparity between the number of nodes deemed active post their discovery-crawl (consistently surpassing 10'000) and the unique nodes responding to a FINDNODE request. As highlighted in the routing table analysis, the count of non-empty routing tables per crawl fluctuates between 11'549 and 12'143. However, when filtered to select a single routing table per node, the figures reduce to a range of 9'054 - 9'448 per crawl, falling short of the initially active nodes. This discrepancy might arise from nodes frequently altering their ENRs. By the time the crawler identifies their latest ENR and queries their routing table, they might have already updated their ENR. Yet, they still register as active during the activity check, responding to the PING but signaling an ENR change.

Another dimension worth exploring is the direct relationship between group size and the number of non-empty routing tables per group, as depicted in figure 4.8. When both metrics align, it indicates that every ENR from that node

responded to the FINDNODE request. It's curious why certain nodes respond regardless of the ENR's recency, while most only react when approached with the latest ENR.

Lastly, understanding the reasons behind the behavior of some nodes—those that disappear from all routing tables shortly after becoming inactive, compared to others that linger for weeks—could offer valuable insights.

Bibliography

[1] P. Wackerow, "The concept of proof of work," 2022, accessed: 2023-08-06. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/

[2] C. Smith, "The concept of proof of stake," 2023, accessed: 2023-08-06. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[3] F. Lange, "Eip-778: Ethereum node records (enr)," 2017, accessed: 2023-08-06. [Online]. Available: https://eips.ethereum.org/EIPS/eip-778

[4] https://github.com/hwwhww, "The specs of the consensus layer," 2023, accessed: 2023-08-06. [Online]. Available: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md

[5] https://github.com/emhane, "The discv5 protocol," 2022, accessed: 2023-08-06. [Online]. Available: https://github.com/ethereum/devp2p/blob/master/discv5/discv5-theory.md

[6] h. https://github.com/raulk, https://github.com/jhiesey, "The kademlia dht specs," 2022, accessed: 2023-08-06. [Online]. Available: https://github.com/libp2p/specs/blob/master/kad-dht/README.md

[7] https://github.com/vyzo, "The specs of the gossipsub protocol," 2020, accessed: 2023-08-06. [Online]. Available: https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md

[8] C. Smith, "Overview of ethereum's networks," 2023, accessed: 2023-08-06. [Online]. Available: https://ethereum.org/en/developers/docs/networks/

[9] https://github.com/AgeManning, "Beacon mainnet specs," 2021, accessed: 2023-08-06. [Online]. Available: https://github.com/eth-clients/eth2-networks/tree/master/shared/mainnet

[10] https://github.com/q9f, "Bepolia testnet specs," 2022, accessed: 2023-08-06. [Online]. Available: https://github.com/eth-clients/sepolia

[11] https://github.com/ardislu, "Prater testnet specs," 2021, accessed: 2023-08-06. [Online]. Available: https://github.com/eth-clients/goerli

[12] https://github.com/zah, "Beacon chain fork versions," 2023, accessed: 2023-08-06. [Online]. Available: https://github.com/eth-clients/eth2-networks/blob/e930d81f7c9db816c88d1a9336be8cef858f7f4d/shared/mainnet/config.yaml