Measuring Ethereum Network Peers

Seoung Kyun Kim
University of Illinois at Urbana-Champaign
skim104@illinois.edu

Zane Ma
University of Illinois at Urbana-Champaign
zanema2@illinois.edu

Siddharth Murali
University of Illinois at Urbana-Champaign
smurali2@illinois.edu

Joshua Mason
University of Illinois at Urbana-Champaign
joshm@illinois.edu

Andrew Miller
University of Illinois at Urbana-Champaign
soc1024@illinois.edu

Michael Bailey
University of Illinois at Urbana-Champaign
mdbailey@illinois.edu

ABSTRACT

Ethereum, the second-largest cryptocurrency valued at a peak of $138 billion in 2018, is a decentralized, Turing-complete computing platform. Although the stability and security of Ethereum—and blockchain systems in general—have been widely-studied, most analysis has focused on application level features of these systems such as cryptographic mining challenges, smart contract semantics, or block mining operators. Little attention has been paid to the underlying peer-to-peer (P2P) networks that are responsible for information propagation and that enable blockchain consensus. In this work, we develop NodeFinder to measure this previously opaque network at scale and illuminate the properties of its nodes. We analyze the Ethereum network from two vantage points: a three-month long view of nodes on the P2P network, and a single day snapshot of the Ethereum Mainnet peers. We uncover a noisy DEVp2p ecosystem in which fewer than half of all nodes contribute to the Ethereum Mainnet. Through a comparison with other previously studied P2P networks including BitTorrent, Gnutella, and Bitcoin, we find that Ethereum differs in both network size and geographical distribution.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Network dynamics**; **Peer-to-peer protocols**; *Peer-to-peer networks*; • **Computer systems organization** → *Peer-to-peer architectures*;

KEYWORDS

Ethereum, DEVp2p, peer-to-peer computing, network measurement

ACM Reference Format:
Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Measuring Ethereum Network Peers. In *2018 Internet Measurement Conference (IMC '18), October 31-November 2, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3278532.3278542

IMC '18, October 31-November 2, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-5619-0/18/10.
https://doi.org/10.1145/3278532.3278542

1 INTRODUCTION

The historic rise of blockchain based cryptocurrencies to over $327 billion in market capitalization [11] has sparked significant research efforts studying their reliability, performance, and security. The central tenet of these cryptocurrencies is their decentralization, which is achieved through blockchain consensus over a peer-to-peer (P2P) network. Bitcoin, the highest valued cryptocurrency, has received the most thorough scrutiny, with many studies analyzing its peer properties and network health [1, 13, 43, 46]. In contrast, the network layer for Ethereum, the second largest cryptocurrency, has gone mostly ignored, even though it employs a completely distinct, more complex P2P protocol.

Ethereum is commonly referred to as a cryptocurrency competitor to Bitcoin, but it is actually much more than a currency. The Ethereum Foundation explains that its token, ether, is not intended to be a currency; it is a byproduct of a much larger vision, a fuel for operating a "world computer" [19, 27, 28]. Ethereum enables decentralized execution of the "world computer" by providing a platform for blockchain-based smart contracts, which are programs that other Ethereum addresses/actors can audit and participate in. For instance, a contract can encode a fair lottery that is auditable by anyone on the network. Execution of a contract is validated by all up-to-date blockchain participants, to ensure correct execution by blockchain consensus. Bitcoin also facilitates the concept of a smart contract, but usage is limited to currency transactions. What differentiates Ethereum from Bitcoin is "the built-in Turing-complete programming language that allows anyone to create contracts for any usage" [8].

The flexibility offered by Ethereum's smart contracts has attracted many users, developers, and investors, leading Ethereum to become the second largest cryptocurrency valued at a peak of 138 billion US dollars [11] in 2018. Ethereum's momentum has prompted many studies at its application layer such as smart contract code analysis [5, 39, 59] or attacks on smart contracts [3, 15]. However, to our knowledge, work exploring its underlying P2P network structure is sparse. Considering its unique capabilities, purpose, and prevalence, we hypothesize that the properties of the Ethereum P2P network are different from those of the Bitcoin network and other previously explored P2P networks, such as BitTorrent and Gnutella. Quantifying Ethereum's P2P network provides insight into its composition and robustness, while also highlighting areas of potential dysfunction.

Before undertaking the measurement of Ethereum network peers, we first performed a case study on the two most popular clients, Geth and Parity, to gain insight into the behavior of Ethereum nodes. We then built NodeFinder, an open-source[1] and ethical measurement tool for characterizing the three protocols that constitute Ethereum's P2P network: RLPx for node discovery, DEVp2p for application session establishment, and Ethereum subprotocol for Ethereum-specific operations. We deployed NodeFinder from April 18–July 8, 2018 and collected two datasets: 1) a comprehensive view of the Ethereum ecosystem via the nodes seen within the full 82 day collection period and 2) a 24 hour snapshot view of the peers on the main Ethereum network to understand the network's instantaneous properties, such as size, latency, geographic distribution, and node freshness. After performing consistency checks as well as external validation, we ultimately discovered 228% more nodes than prior work.

Our investigation reveals that Ethereum operates on top of a noisy P2P network, which consists of a multitude of different protocols, subprotocol networks, and blockchains. In fact, we find that fewer than half of DEVp2p nodes contribute to the main Ethereum blockchain. Even amongst productive Ethereum nodes, we discover a jumble of both official and unofficial clients running a wide range of stable and unstable versions. Perhaps most surprisingly, we observe that the two most popular clients have a fundamental difference in their RLPx implementations that has the potential to cause friction between significant portions of the overall network. Finally, in our comparison to other popular P2P networks such as Gnutella, BitTorrent, and Bitcoin, we find that Ethereum has differences in both network size and distribution, which presents unique challenges for future growth and improvement.

In summary, the contributions of this paper are as follows:

- The development of NodeFinder, a new open-source tool for scanning and monitoring Ethereum's P2P network that discovers over 10K more nodes than existing efforts.
- A comprehensive exploration of Ethereum's diverse P2P ecosystem that exhibits many hallmarks of a noisy, inefficient network.
- A snapshot examination of Ethereum's main network and a comparison with other well-known P2P networks, such as Gnutella, BitTorrent, and Bitcoin.

2 BACKGROUND

Ethereum is a blockchain platform for distributed computing and was the first major blockchain to support Turing-complete scripting via smart contracts, which are expressed as opcodes specific to the Ethereum Virtual Machine (EVM) [55]. Ethereum also operates as a cryptocurrency by supporting the transfer of tokens called *ether* between Ethereum accounts. In addition to its use as a fiat currency, ether is also used to incentivize Ethereum nodes to perform distributed computation. Ether can also be converted to *gas*, which is used as a fee for executing transactions and smart contracts, and effectively mitigates spam.

Similar to other cryptocurrencies, Ethereum's blockchain is managed by a peer-to-peer (P2P) network. Ethereum's network communication is comprised of three different protocols, which run on top of UDP and TCP: *RLPx* for node discovery and secure transport, *DEVp2p* for application session establishment, and the *Ethereum application-level protocol* (henceforth referred to as Ethereum subprotocol). We provide a high level overview in Figure 1 and expand on the pertinent details for each protocol below, drawing upon official specifications [9, 35, 38].

[1] https://www.github.com/teamnsrg/ethereum-p2p

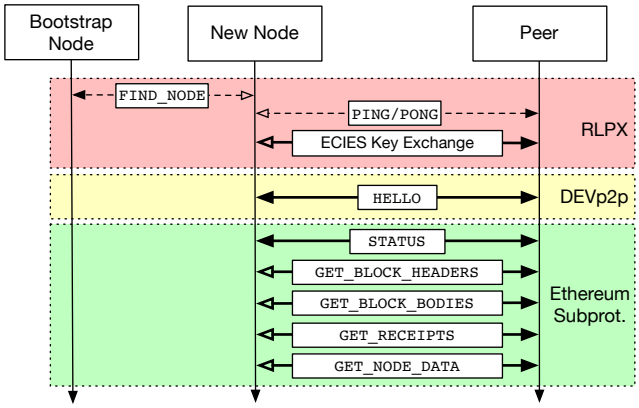

Figure 1: Ethereum Network Protocols—We provide an overview of RLPx, DEVp2p, and Ethereum subprotocol and display a typical workflow for a new node joining the Ethereum network. These protocols run on top of UDP (dotted-line) and TCP (solid line), through a series of request (solid arrow) and response (hollow arrow) messages.

2.1 RLPx

RLPx implements node discovery based on the routing algorithm of Kademlia, a widely used distributed hash table (DHT) [41], to build an efficient network with a topology of low diameter [36, 38, 56]. We first discuss how Kademlia operates and then highlight the unique features of RLPx.

Kademlia is a UDP-based protocol for distributed nodes to store and retrieve data. Node IDs (randomly generated) and keys for stored data (generated by hashing the data) are both represented as 160-bit values, which allows for direct comparison between node IDs and data keys. More specifically, Kademlia uses bitwise XOR to compute a distance $d(a, b) = a \oplus b$ between two 160-bit values and then, using the integer value of this *bitwise XOR distance*, stores data at nodes with node IDs that are close to the data's key. Kademlia's operation hinges upon this local, deterministic mapping between data and network nodes to locate nodes quickly.

Each Kademlia node maintains a routing table for monitoring peer-connected nodes and determining which neighbors store the data for a given key. The routing table is split into 160 buckets based on the XOR distance between a node's own node ID and the neighboring node ID. Each bucket is a list of nodes that are between 2^i to 2^{i+1} in XOR distance. Each bucket list is limited to a maximum of k nodes and is thus called a *k-bucket*. When a new neighboring node is detected, Kademlia adds the node to the appropriate k-bucket. However, if the target k-bucket is full, Kademlia's eviction policy favors old nodes and only adds a new

node if the least recently active pre-existing node is not lively, i.e., it does not respond to a PING message with a PONG message.

In order for a newly instantiated Kademlia node to find peers, it first adds a hard-coded set of bootstrap node IDs to its routing table. Subsequently, when attempting to locate a target node on the network, the node searches its routing table for the α (typically three) peers that are closest to the target node. The node then sends these α nodes a FIND_NODE message that specifies the target node ID, and each peer responds with the list of k nodes from its own routing table that are closest to the target. The querying node adds any new node information (i.e., node ID, IP Address, UDP/TCP ports) it discovers through this process to its routing table, and then iteratively repeats the process until it converges on the target node.

There are five primary differences between RLPx and Kademlia. First, RLPx does not support data storage/retrieval—it only supports node discovery and routing. Second, RLPx uses 512-bit node IDs instead of 160-bit node IDs. Third, node IDs also function as public keys and are used in RLPx to ultimately establish an authenticated TCP connection that provides security features including signed packets and encryption after a Elliptic Curve Integrated Encryption Scheme (ECIES) key exchange. Fourth, RLPx does not calculate XOR distance directly on node IDs; instead, it performs distance calculation on the Keccak-256 hash [6] of the node ID. Finally, RLPx uses the floor of $log_2(a \oplus b)$ as its distance metric, which corresponds to 257 distinct node buckets.

2.2 DEVp2p

After peer nodes have been discovered through RLPx and a secure TCP connection is established, DEVp2p negotiates an application session between two connected peers. Each node must first send its peer a HELLO message, which details its own node ID, DEVp2p version, client name, supported application protocols/versions, and the port number (30303 by default) that the node is listening on[2]. Based on HELLO message information, the nodes may begin to transmit application data packets over DEVp2p. During periods of inactivity, DEVp2p nodes will periodically send DEVp2p PING messages (not to be confused with RLPx PING) at an interval set by the client to ensure their connected peers are still active and have not crashed. If a corresponding DEVp2p PONG message is not received within the maximum allowed idle time set by client, then the node will send a DISCONNECT message, which may include an error code explaining the disconnect.

2.3 Ethereum Subprotocol

The Ethereum subprotocol runs on top of DEVp2p and is denoted as 'eth' during DEVp2p HELLO exchange. At a high level, the Ethereum subprotocol is used to retrieve and store information on the Ethereum blockchain. Note that this does not include the details of smart contract execution and Ethereum blockchain mining—rather, we focus on the messages used to communicate blockchain state information over the network. The description below applies to version 62/63 of the Ethereum subprotocol, which was added in October 2015 [30] and is the most recent version as of 2018.

The first message that must be sent by both peer nodes after the DEVp2p HELLO handshake is a STATUS message, which conveys the current state of a node's blockchain. It contains a node's protocol version, network ID (multiple distinct Ethereum networks exist), and the Keccak-256 hash of the first block in the blockchain, a.k.a. the *genesis hash*, since there may also be multiple distinct blockchains for a single network ID. The mainstream Ethereum blockchain exists on network ID 1 (i.e., Mainnet) with genesis hash d4e56740...b1cb8fa3, and it supports the DAO fork[3]. STATUS information is used by nodes to determine which peers they should connect to. Similar to DEVp2p, if a node encounters an Ethereum peer that is on a different Ethereum network or genesis hash, it will disconnect from that peer.

Peers that remain connected after STATUS message exchange utilize two STATUS message fields to coordinate blockchain synchronization: the hash of the most recent block known to a node (i.e., best hash) and the total difficulty of its blockchain. For illustrative purposes, consider a new node joining the Ethereum network. It begins downloading a local copy of the full blockchain by first sending GET_BLOCK_HEADERS messages to obtain a list of block headers, which include block meta information such as parent block hash, miner address, and a free-form field for extra information, which is used to detect the DAO fork and distinguish between mainstream Ethereum and Ethereum classic, amongst other uses. After it has compiled a list of missing block hashes, the node then sends GET_BLOCK_BODIES messages to retrieve full block contents and verify the validity of the blockchain.

There are two forms of Ethereum blockchain validation: 1) block header validation and 2) blockchain state validation. Block header validation, as defined in Section 4.3.4 of the Ethereum Yellow Paper [55], checks a block's parent block hash, block number, timestamp, difficulty, gas limit, and valid proof-of-work hash. Blockchain state validation consists of sequentially executing all transactions, recording every account's state in a global database, and inserting each state snapshot as a node in the global Merkle Patricia state tree. Blockchain state validation requires significantly more computation and time than block header validation.

In order to reduce the time for new nodes to synchronize and validate the entire blockchain, Ethereum version 63 introduced fast sync, an optional operational mode which reduces the blockchain state validation workload and improves syncing times by approximately an order of magnitude [54]. After downloading all block headers and bodies, a fast sync node picks a *pivot point* block that is close to the most recent head of the blockchain. From the genesis block to the pivot point, the node performs fast block header validation via GET_RECEIPTS messages, which retrieve meta information including gas consumption, transaction logs, and status code. At the pivot point, a fast sync node utilizes GET_NODE_DATA messages to download a global state database at that block. From the pivot point onward, the node performs full blockchain validation.

[2]In practice, the "listenPort" field is unused/ignored by most clients since the RLPx TCP connection already exists on a port, and port information is de facto obtained at the RLPx layer

[3]The DAO fork is a hard fork that occurred on July 20th, 2016 returning approximately $40 million worth of Ether stolen from the DAO contract in June 2016 to a refund smart contract [14]. As a result of the fork, the mainstream Ethereum blockchain split into two, and the non-supporting blockchain became Ethereum Classic.

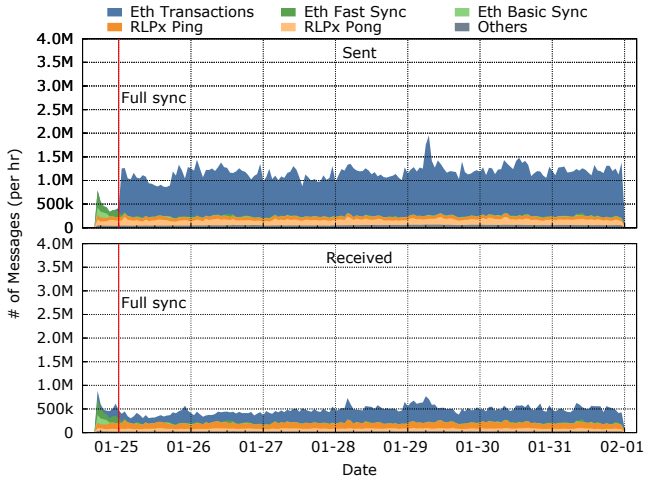

Figure 2: Geth Message I/O—The counts of messages sent/received by Geth reveal that an overwhelming 77.4% and 53.5% of sent/received messages were `TRANSACTIONS` messages.

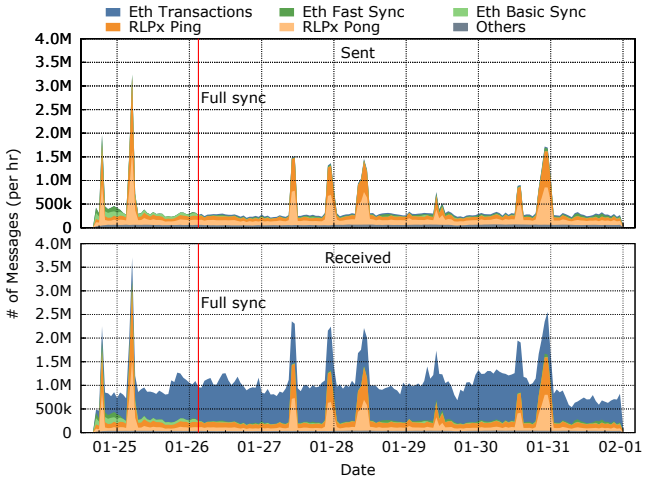

Figure 3: Parity Message I/O—The counts of messages sent/received by Parity reveal that Parity sends significantly fewer `TRANSACTIONS` messages than Geth, only 4.6% of sent messages. Parity also experiences irregular RLPx `PING/PONG` spikes.

Once a node has been synced to the blockchain, it can actively participate in the network by announcing and listening to two possible blockchain events: new transactions and new blocks. In order to add new transaction(s) to the blockchain, an Ethereum node (hereafter referred to as the "transaction origin node") can broadcast a `TRANSACTIONS` message to all of its active Ethereum peers. For non-origin nodes, upon receipt of a `TRANSACTIONS` message, all transactions in the message are validated locally to ensure that they are signed properly, do not exceed size/gas limits, do not transact a negative value, and have senders with sufficient Ether/gas. The recipient node then broadcasts valid transactions to all peers except those that are likely to already know about the transaction, i.e., the peers that sent the transaction and the peers that have previously been sent the transaction. New block propagation occurs similarly through `NEW_BLOCK_HASHES` and `NEW_BLOCK` messages.

3 CASE STUDY

While the documentation for Ethereum's full network stack (RLPx, DEVp2p, Ethereum subprotocol) defines message formats and functionality, it provides sparse guidance on message sending behavior, which can be implemented in a variety of ways. For instance, when a new node downloads an initial copy of the blockchain using `GET_BLOCK_HEADERS` and `GET_BLOCK_BODIES`, the order and concurrency of downloads is open to implementation interpretation, and this choice can have drastically different impacts on the network. Thus, to inform the design of NodeFinder, an Ethereum network measurement tool, we deployed and profiled the operation of the two most popular Ethereum clients [22], Go Ethereum a.k.a. Geth (Ethereum's official Golang implementation) [29], and Parity, an unofficial Rust implementation [47].

From January 24 – January 31, 2018, we instrumented and ran Geth version 1.7.3 and Parity version 1.7.9 with default settings on two separate Ubuntu 16.04 machines, each with 128GB RAM, 32 cores, and a 10 Gb/s network link. We recorded all messages

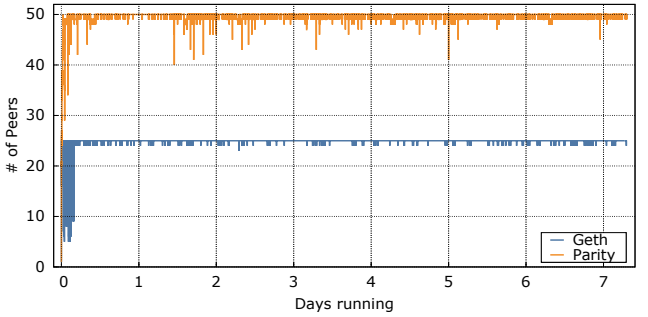

Figure 4: Peer Counts—Geth and Parity converge to their default 25 and 50 peers, respectively, and remain relatively stable over the course of a week.

sent and received (Figures 2 and 3), along with any changes in the number of connected peers (Figure 4).

From our case study, we made the following observations:

(1) Geth and Parity reached their default peer limits in a matter of minutes, with Geth converging to a maximum of 25 peers, and Parity converging to 50 peers (Figure 4). A network-encompassing scanner must ignore the hardcoded peer limits to continuously discover new nodes and monitor existing ones. Additionally, even though Geth and Parity nodes were relatively stable—they were at maximum peer occupancy 99.1% and 91.5% of the time, respectively—they still fluctuated enough to provide brief windows of opportunity for a network scanner to connect to them over time.

(2) Once a node has synchronized with the blockchain, `TRANSACTIONS` messages dominate network I/O. Geth and Parity received similar proportions of `TRANSACTIONS`

Disconnect Msg	Received		Sent	
	Geth	Parity	Geth	Parity
Too many peers	3,938 (72.55%)	113,014 (95.19%)	2,073,995 (99.59%)	1,493,488 (88.58%)
Subprotocol error	433 (7.98%)	174 (0.15%)	3,856 (0.19%)	–
Disconnect requested	967 (17.82%)	2,741 (2.31%)	2,730 (0.13%)	9,322 (0.55%)
Useless peer	41 (0.76%)	108 (0.09%)	1,859 (0.09%)	168,341 (9.98%)
Already connected	31 (0.57%)	2,681 (2.26%)	124 (0.01%)	124 (0.01%)
Read timeout	15 (0.28%)	10 (0.01%)	24 (0.00%)	14,780 (0.88%)
Client quitting	3 (0.06%)	1 (0.00%)	3 (0.00%)	–
Total	5,428 (100%)	118,729 (100%)	2,082,591 (100%)	1,686,055 (100%)

Table 1: Disconnect Reasons—The majority of `DISCONNECT`s, both sent and received, are due to peers that have reached their maximum peer limit. Significantly more `DISCONNECT`s sent than received suggests a large number of incoming peer connections.

messages, but Geth sent significantly more to the network. Examining the source code reveals that while Geth broadcast transactions to all of its peers, Parity only sent transactions to $\sqrt{n}$ peers. Transaction broadcast can quickly become a crippling bottleneck for an Ethereum network monitor that connects to every peer it finds, due to the large numbers of duplicate messages that every peer would send the monitor.

(3) To better design a network wide Ethereum scanner that uncovers as many nodes as possible, we examined the `DISCONNECT` messages (Table 1). Intuitively, an effective scanner would avoid all disconnect scenarios. The most popular disconnect reason for both Geth and Parity was *Too many peers*, which occurs when connecting to a node that has already reached its maximum peer limit. Our scanner should accept all incoming connections and never send out *Too many peers* disconnects. Our scanner will also attempt to overcome the *Too many peers* messages it receives by slowly and deliberately re-attempting to connect to nodes at max peer capacity. The next most commonly observed disconnect reasons were *Subprotocol error*, *Disconnect requested*, and *Useless peer*. These indicate either a non-Ethereum protocol on the DEVp2p network, an incompatible Ethereum blockchain (e.g., Ethereum Classic), or a faulty protocol implementation. Because we ultimately aim to measure the functional Ethereum network (i.e., the main network with correct genesis hash and DAO fork), we do not try to reduce these disconnections.

(4) Interestingly, even though the case study Geth instance had fewer concurrent peers than Parity, it received and sent significantly more `DISCONNECT` messages due to *Subprotocol error*. Parity sending zero *Subprotocol error* messages is not surprising as the client considers any error code beyond `0x0b` as "Unknown" and does not implement sending a disconnect message to peers causing the error. This however does not explain why Parity receives relatively smaller number of such errors from its peers compared to Geth. Our hypothesis is that the Parity case study node might have peered primarily with other Parity nodes, which do not send *Subprotocol error* messages. One possible mechanism is through our discovery of Parity's incorrect implementation of the XOR distance metric as discussed in Section 6.3.

4 DEVP2P NODEFINDER

Although RLPx and DEVp2p were developed specifically for Ethereum, DEVp2p in particular was designed to support any number of higher level application subprotocols. In order to inspect the full DEVp2p ecosystem underlying Ethereum, we built NodeFinder, an open source Ethereum monitoring tool that identifies active DEVp2p nodes and periodically retrieves their client information (DEVp2p `HELLO`) as well as their Ethereum blockchain status (Ethereum `STATUS`). We designed the scanner following the observations made in Section 3. NodeFinder is based on Geth version 1.7.3, which remains compatible with all current versions of RLPx, DEVp2p, and Ethereum subprotocol. Below, we describe several modifications made to the base Geth implementation to attain broad coverage of all DEVp2p nodes.

First, NodeFinder ignores the maximum peer limit. A normal Geth client relies on node discovery and incoming connections to populate its peer table. The discovery process is initiated whenever the client has room for more peers and has tried connecting to all nodes from its most recent round of discovery. The client accepts incoming connections until the maximum peer limit is reached. If the limit is exceeded, either through discovery or incoming connections, the client terminates any pending peer connections by sending out *Too many peers* disconnects. NodeFinder ignores the maximum peer limit at both the DEVp2p and Ethereum layers in order to continuously perform discovery and minimize peer disconnects.

Second, NodeFinder disconnects from peers once it has checked for the DAO fork block via `GET_BLOCK_HEADERS`, which follows immediately after a successful Ethereum handshake. A normal Geth client maintains its peer connections as long as possible—until the connection fails or either side of the connection requests to disconnect—because it is effectively a file-sharing client designed to continuously download the blockchain, which never stops growing. Considering the tremendous amount of traffic generated by Ethereum connections (as shown in the case study), maintaining all peer connections indefinitely and handling every message, while ignoring the maximum peer limit, is impractical. NodeFinder disconnects from peers and frees up their peer slots as soon as it is done collecting information from peer connection establishment, which consists of DEVp2p handshake, Ethereum handshake, and DAO fork block verification (i.e., requesting the DAO fork block

header). NodeFinder then periodically reconnects to known nodes to track longitudinal properties such as liveliness and churn. Involving only 3 message exchanges at most, NodeFinder occupies peer slots for less than a second in most cases, and no more than 2 minutes in the worst case where both endpoints take the maximum allowed times to read (30s) and write (20s) messages.

To periodically re-connect to previously seen peers, we modified Geth's outgoing discovery mechanism so that when NodeFinder successfully completes "dynamic-dials,"—dials to new nodes learned from node discovery—the dialed addresses are automatically added to a `StaticNodes` list and re-dialed as "static-dials" every 30 *minutes*. As the `StaticNodes` list is expected to grow large over time, NodeFinder schedules static-dials, dynamic-dials, and node discovery in separate queues to prevent the static-dials from delaying other tasks. Up to 16 concurrent dynamic-dials (the default limit determined by Geth's `maxActiveDialTasks` constant) and a single node discovery are handled concurrently. Every static-dial is handled immediately with no concurrency limit. We store all the addresses and their last-dialed timestamps to a local database to allow NodeFinder to re-generate the most recent `StaticNodes` list in case it restarts due to errors. We removed the addresses resolution step because resolving every failed address in the list generates excessive RLPx node discovery traffic and may negatively affect the network. Instead, we remove stale addresses with a last successful TCP connection time of more than 24 hours from the list. Bootstrap nodes—which normal clients use as fallback nodes when it has no connected peers—are added to the `StaticNodes` list and periodically re-dialed like any other nodes.

Lastly, for both debugging and data collection purposes, we co-opted Geth's built-in logging mechanism to record information. When peers send `HELLO`, `DISCONNECT`, and `STATUS` messages, NodeFinder logs decoded content of each message in separate lines. If DAO fork block verification occurs, whether the peer supports or opposes the fork is also logged separately. Every log message resulting from a peer connection is prepended with following information: timestamp (in Unix time with microsecond-precision), peer's node ID, IP address, port, connection type (dynamic-dial, static-dial, or incoming), connection latency, and duration of the connection. To estimate the peer's latency, NodeFinder obtains the smoothed round-trip time of the connection directly from its underlying TCP socket every time it sends or receives a message.

Geth's DEVp2p server has several hardcoded constants that determine its discovery and connection rates. Brief descriptions and default values of the constants are listed below:

- `lookupInterval` (4s) - minimum time interval between node discoveries (based on start time)
- `defaultDialTimeout` (15s) - TCP dial timeout
- `frameReadTimeout` (30s) - TCP read timeout
- `frameWriteTimeout` (20s) - TCP write timeout
- `maxActiveDialTasks` (16) - maximum number of concurrently dialing connections
- `maxAcceptConns` (50) - maximum number of concurrently accepting connections

Although changing these constants would most likely improve node discovery connection rates and result in more efficient handling of idle connections, we choose to leave them as their default values to minimize our measurement impact on the network.

5 MEASUREMENTS

From April 18–July 8, 2018, we instrumented and ran 30 instances of NodeFinder on an Ubuntu 16.04 machine with 128GB RAM, 32 cores, and a 10 Gb/s network link. To handle a large number of concurrent peer connections, we increased the system limits on number of open files to 1,048,576 and configured network buffer size and ephemeral port range of TCP as listed below:

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65000
```

We note that the measurement nodes' performance gradually degraded from around the 3rd week of May to May 28th due to substantial delays in query results from their central database that kept track of status of scanned targets. We resolved the issue by removing unnecessary database indexes and simplifying query statements. Sharp drops on the 28th present in time series graphs in this paper are due to 13-hour downtime that resulted from the structural change of the database. We resumed the measurement at 1pm (CST) of the same day. After the change, the measurement nodes remained stable until the end, from May 29th to July 8th.

5.1 Measurement Ethics

In accordance with the ethics guidelines presented by the Menlo Report [4], we made conscious efforts to reduce or eliminate the harm done by our measurements on the Ethereum network. We also followed many of the best practices outlined by ZMap [18] and ensured that all scanning nodes had DNS names that signal benign scanning, as well as a website on port 80 explaining the purpose of our scanning. We received no requests to be excluded from our Ethereum measurements. Further, we only sent well-formed, standard compliant messages and conformed to normal Ethereum peer behaviour, when possible. Lastly, we attempted to minimize the number of NodeFinder nodes on the network to avoid unintentional sybil attacks.

5.2 Internal Validation

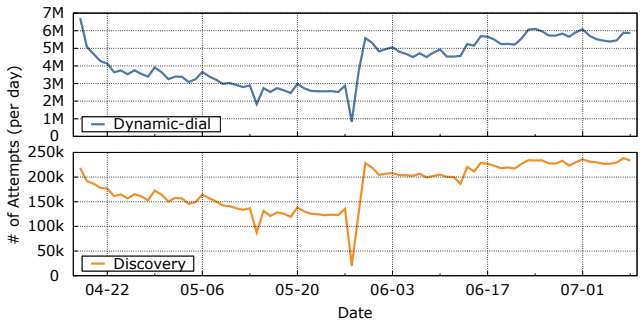

Figure 5: Discovery and Dynamic-dial Attempts—NodeFinder instances made 219,180 discovery and 5,328,144 dynamic-dial attempts per day on average during the stable period. The ratio between the two remains visibly constant.

To validate the consistency of our measurement approach and results, we observe the 30 NodeFinder instances' node discovery and connection rates. We first confirm that they continuously performed node discovery throughout the measurement, with each node making about 304 attempts per hour on average during the stable period (Figure 5). Given that the normal Geth client from our case study made 180 attempts per hour, we find the NodeFinder's discovery attempt rate acceptable as it is clearly faster than a normal client's while not exceeding the 4-second interval limit enforced by `lookupInterval`. We also observe that the nodes' dynamic-dial attempt rate remained proportional to the discovery attempt rate at a constant factor, as expected, since dynamic-dials always originate from node discovery results.

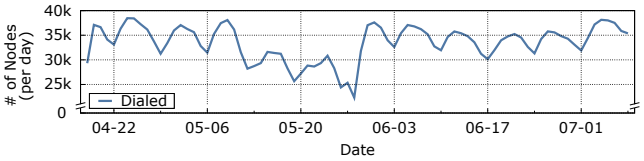

Figure 6: Number of Nodes Dynamic-dialed—NodeFinder instances attempted dynamic-dials to 34,730 unique nodes per day on average during the stable period.

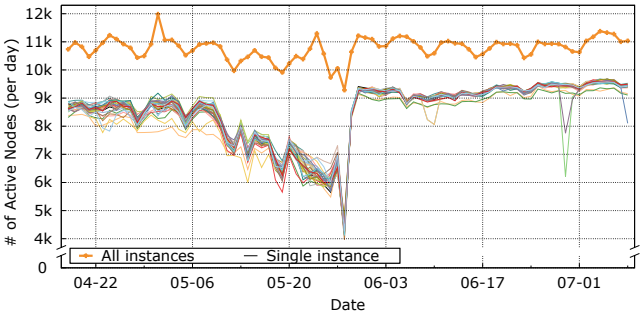

Figure 7: Number of Nodes Responded to Dynamic-dials—10,919 unique nodes responded to dynamic-dials per day on average during the stable period. The daily number of responding nodes observed by the NodeFinder instances as a whole remains relatively consistent, regardless of each instance's stability.

Next, in Figure 6 and Figure 7, we find the NodeFinder instances attempted dynamic-dials to 34K unique nodes and received either `HELLO` or `DISCONNECT` messages back from nearly 11K nodes per day on average during the stable period, showing about 31.4% chance of finding an active DEVp2p node. The figures also show that the number of unique nodes observed by all 30 instances as a whole remained relatively consistent, even throughout the unstable period in which each instance's rates clearly declined. While this indicates that running 30 instances of the tool was sufficient to overcome each instance's performance decline, the overall network coverage remains unknown. Discovering a total of 455,641 nodes over the 82-day period at a consistent rate of over 5K per day, the 30 instances showed no clear sign of approaching full network coverage. We hypothesize that the unexpectedly high growth rate is caused by nodes that constantly generate new node IDs. We further discuss the growth rate in Section 5.4.

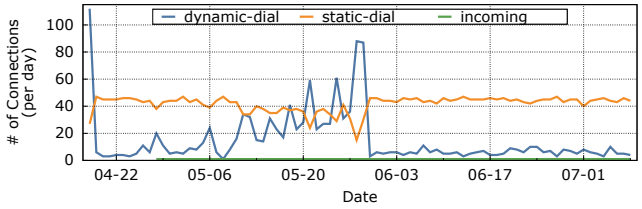

Figure 8: Connections from/to Bootstrap Node—Both NodeFinder instance and the bootstrap node frequently find each other through node discovery. The bootstrap node is periodically re-dialed.

Lastly, we look at the number of connections made between a NodeFinder instance and a known bootstrap node[4]to verify if NodeFinder static-dials discovered nodes as designed. Figure 8 shows that the NodeFinder instance connected to the bootstrap node through about 6 dynamic-dials and 44 static-dials per day on average during the stable period. The number of static-dials to a single node is expected to be no greater than 48 per day because NodeFinder's static-dial interval is set to 30 minutes. The observed number is slightly lower than the maximum because NodeFinder reschedules next static-dial upon completion of *any type* of outbound connection attempt, i.e., an unscheduled regular dial pushes back next re-dial time. The noticeable spikes in number of dynamic-dials during the unstable period are in response to a lack of static-dials resulted from the instances' performance issue.

To confirm that NodeFinder was able to find newly-joined nodes within a reasonable time frame, we observe how long the 30 NodeFinder instances—which all started and joined the DEVp2p network at the same time—took to find and make a successful DEVp2p connection (i.e., exchanged either `HELLO` or `DISCONNECT` messages) with each other. Each instance discovered the other 29 nodes in less than 9 hours, the fastest completion time being a little over 3 hours. This provides us confidence that the NodeFinder should be able to find any node (participating in node discovery) on the network.

5.3 External Validation

To validate the coverage of NodeFinder with an external data source, we compare our results with ethernodes.org, an independent website that reports the estimated size of Ethereum networks and details of their nodes based on information collected with one or more crawling nodes through both outgoing and incoming connections [22, 50]. The list of nodes and their details are collected from the website's Mainnet nodes page, which lists all nodes that have been seen using network ID 1 within 24 hours. As Ethereum networks are identified based on both network ID and genesis hash, we look at each node's reported genesis hash and consider only those with the Mainnet blockchain's genesis hash (`d4e567...cb8fa3`) for our comparison. We then compare the filtered set of nodes against

[4] `enode://78de8a0916848093...@191.235.84.50:30303`

our own dataset of scanned nodes that are part of the Mainnet. Both datasets, collected at midnight of April 24th in Central Standard Time, are based on scans run over a 24-hour period beginning on April 23rd.

Source	EN	NF	NFR	NFU
EN	4,717	3,856	2,620	1,236
NF	—	16,831	5,951	10,880

Table 2: Number of Nodes in Intersections of NodeFinder and Ethernodes Sets—EN: Ethernodes, NF: NodeFinder, NFR: NodeFinder Reachable Nodes, NFU: NodeFinder Unreachable Nodes

We find that only 4,717 out of 20,437 nodes listed on Ethernodes' Mainnet nodes page at the midnight actually operated on the Mainnet blockchain. Our comparison, summarized in Table 2, shows that the 30 NodeFinder instances were able to pick up 12,114 more Mainnet nodes. NodeFinder and Ethernodes overlap at 3,856 common nodes, or 81.8% of all nodes found by Ethernodes. To determine if we have ever learned about the missing 891 nodes, we looked at larger datasets: 1) a list of all Ethereum nodes seen during the same period, and 2) a similar list including all DEVp2p nodes. With 8 and 359 matches from the first and second list respectively, we speculate that our scans successfully detected 359 of the missing nodes during the 24-hour period but were unable to verify their network due to failing to receive their `STATUS` messages. A closer look at the missing nodes' supported protocols shows that 61 of them were running light nodes—which run different protocols, namely Light Ethereum Subprotocol (LES) and Parity Light Protocol (PIP), for simply accessing and verifying a small portion of an Ethereum blockchain [10]. As NodeFinder is not designed to support the light protocols, it is unable to exchange messages with them to obtain their network information. Looking at the nodes' client information, we also find that 170 of the missing nodes were running Parity 1.6.8/1.7.0-beta-windows-msvc but the root cause of their absence from our scans remains unknown. Because they amount only 1.4% compared to the 12K nodes missing in Ethernodes' dataset, we consider this a minor limitation.

To understand how the NodeFinder instances fare against Ethernodes in network coverage, we compare how many publicly unreachable nodes are observed in each dataset. In our dataset, we consider a node publicly reachable if there was at least one successful peer connection started by our nodes during the 24-hour period. As we are unable to obtain such information for the Ethernodes' dataset, we look at its overlap with our reachable and unreachable subsets. Table 2 shows that our method found more nodes than Ethernodes in both categories. Further, the difference of almost one order of magnitude in the number of publicly unreachable nodes indicates that the 30 NodeFinder instances have propagated their node addresses throughout a significantly larger portion of the Mainnet than Ethernodes' crawlers have.

5.4 Data Sanitization

To narrow down the source of the unexpected growth rate, we look at the distribution of nodes by IP addresses and find that more than 15% of nodes resided at only 5 IPs. We further look into the IP with the largest number of nodes, 149.129.129.190, and find that all 42,237 nodes from the address ran same client `ethereumjs-devp2p/v1.0.0`[5], and their reported best block hashes were always same as the Ethereum Mainnet genesis block hash. 80% of them were seen only once; the longest time a node remained active on this IP was less than 30 minutes. We categorize these nodes, as well as 4K nodes from 2 other IPs, as anomalies that should not be considered for subsequent analysis as part of the network. Based on our observation of over 20K similarly behaved nodes, we establish the following steps to identify short-lived nodes from IP addresses that abusively generate new nodes (node IDs):

(1) Choose nodes (based on node IDs) that have been active for less than 30 minutes.
(2) Group the chosen nodes by their IPs.
(3) Exclude IPs that map to less than 3 nodes.
(4) Calculate new node generation rate for each IP.
(5) Choose IPs that generate new nodes every 30 minutes or faster on average.

Based on these criteria, we mark 97,930 nodes (21.5% of all nodes) belonging to 1,256 IPs (0.3% of all IPs) as abusive and removed them from the dataset so that our results represent a more accurate estimate of the ecosystem. For the same reason, we also exclude 242 nodes that ran NodeFinder during our data collection period, including our own 37 measurement nodes.

5.5 Limitations

Our analysis results are subject to limitations due to lack of ground truth data and the dynamic nature of the P2P network. Without ground truth data, we are unable to validate our findings and provide an accurate estimate of our coverage of the network. The large amount of churn in the network and presence of publicly unreachable nodes make our network analysis more difficult. Because we partially rely on incoming connections from nodes, we may never be able to have peer connections with some unreachable nodes that connect to a set of selected nodes added to their `StaticNodes` list. Our understanding of the network properties are also limited our analysis does not recover the network topology.

6 PEER ECOSYSTEM

Ethereum operates on an overlay network that consists of RLPx over UDP/TCP, DEVp2p, and Ethereum subprotocol (Figure 1). We performed a comprehensive measurement of all three layers of this ecosystem and found a potpourri of nodes running different protocols and contributing to different blockchains. Even after narrowing down the peer network to non-Classic Ethereum Mainnet nodes, we observed a heterogeneous collection of client types and versions. We also find evidence that the two most popular client types, Geth and Parity, do not interoperate efficiently due to implementation differences. We expand on each section below.

[5] `ethereumjs-devp2p` is a JavaScript implementation of DEVp2p and Ethereum protocol. v1.0.0 is the first version publicly released more than a year prior to the first main release and is known to have issues that cause unstable DEVp2p and Ethereum connections [21].

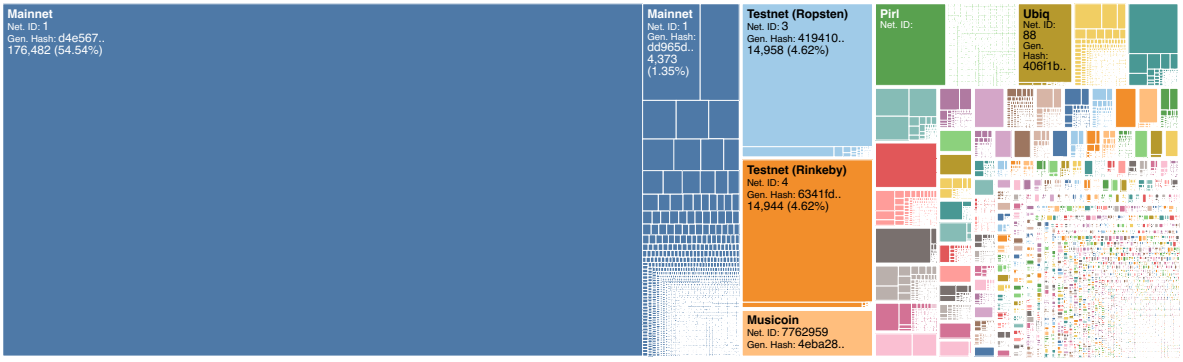

Figure 9: Alternate Ethereum Blockchains—Ethereum Mainnet with the proper genesis hash accounts for only 54.5% of Ethereum nodes, which operate a total of 4,076 networks and 18,829 genesis hashes. The major subdivisions represent different networks, and the minor subdivisions represent distinct genesis block hashes within a network.

6.1 Non-productive Peers

In total, from April 18 to July 8, we found 3,023,275 unique node IDs through RLPx discovery. Of these, we were able to establish an RLPx encrypted TCP connection with 357,710 nodes and successfully exchange DEVp2p `HELLO` messages with 356,492 nodes. We characterize these nodes and discover that 48.2% are useless peers that do not contribute to the Ethereum network because they either do not run the Ethereum subprotocol, or they do not operate on the main Ethereum blockchain.

Service (protocol)	Count	Percentage
Ethereum (eth)	335,036	93.98%
Swarm (bzz)	6,579	1.85%
LES (les)	4,431	1.24%
Expanse (exp)	1,800	0.50%
Istanbul BFT (istanbul)	1,647	0.46%
Whisper (shh)	1,622	0.45%
DubaiCoin (dbix)	1,010	0.28%
PIP (pip)	945	0.27%
MOAC (mc)	583	0.16%
Elementrem (ele)	286	0.08%
Unknown	30	0.01%
30 Others	2,523	0.71%

Table 3: DEVp2p Services—Ethereum subprotocol is the primary protocol found on DEVp2p at 93.98% of the peer network.

DEVp2p is designed to support any number of application level subprotocols that are specified as *capabilities* in the DEVp2p `HELLO` message. In practice, we observe that Ethereum is the predominant service utilizing DEVp2p, accounting for 94.0% of all nodes (Table 3). Non-Ethereum services fall into one of two categories: complementary services that fulfill the "world computer" vision of the Ethereum foundation [27] and competitive services that operate their own protocol and blockchain rather than utilize the Ethereum blockchain as their underlying computational platform. Complementary protocols on the DEVp2p peer network include the Swarm decentralized storage service (6,579 peers), the Whisper anonymized communication protocol (1,622 peers), and light client protocols (5,376 peers)—such as LES, Light Ethereum Subprotocol, and PIP, Parity Light Protocol—that only validate subsections of the Ethereum blockchain. Competing protocols include Istanbul BFT [58], DubaiCoin [2], MOAC Mother Of All Coins [44], and 32 others, which combined constitute 7,849 peers and 2.2% of the overall DEVp2p network.

Within the Ethereum subprotocol, peers can be configured to operate on different Ethereum networks and blockchains via the `networkID` and `genesisHash` (i.e., hash of the genesis/first block) fields of the `STATUS` message. From Ethereum `STATUS` messages received from 323,584 nodes, we found a wide and substantial range of 4,076 networks and 18,829 genesis hashes (Figure 9). The Ethereum blockchain underlying the second most valuable cryptocurrency exists on network 1 (i.e., Mainnet) at genesis hash `d4e567...cb8fa3`—only 176,482 (54.5%) of Ethereum peers had this configuration. We further distinguished these peers by checking for their DAO fork block and found that 3,386 were Ethereum Classic nodes, yielding a total of 173,096 useful Ethereum peers. 97,074 peers were definitively non-Classic peers, and the other 76,022 did not provide a definitive response likely because they had not yet synchronized up to the DAO fork block.

After Mainnet, the largest networks were Ropsten (4.6%) and Rinkeby (4.6%), two official test networks, followed by a host of alternative cryptocurrencies including Musicoin (1.5%), Pirl (1.5%), and Ubiq (1.1%). The long tail of networks represents both the popularity of the Ethereum P2P stack as a base for new alternative blockchains as well as instances of unexpected configurations. For instance, 1,402 networks were only observed for a single peer, and we also observed 10,497 instances of a non-Mainnet peer advertising the Mainnet genesis hash across 1,459 networks. While some of these nodes and networks may be the result of misconfiguration, we cannot confidently attest to the intent behind these use cases. This remains an area for future work.

By design, the peer-to-peer network underlying Ethereum (i.e., RLPx and DEVp2p) supports multiple coexisting protocols and blockchain networks. This is in contrast with other cryptocurrencies such as Bitcoin, Litecoin, and Monero, which define specific ports to provide network-level isolation for different main

networks and test networks. In effect, clients operating on these cryptocurrency P2P networks rarely connect to useless peers. By recording peer information for Ethereum's overlay P2P network, we find an assortment of non-Ethereum protocols and competing blockchains that in total account for 48.2% of the overall P2P ecosystem and can potentially generate significant network noise for non-Classic Ethereum Mainnet clients. This is a direct consequence of Ethereum's flexible design and application-level isolation for distinct services and blockchains.

6.2 Client Heterogeneity

After narrowing down to non-Classic Mainnet Ethereum peers, we examine the client implementations being used on the network. In total, Geth, the official Go client, accounts for 76.6% of all Ethereum peers, followed by Parity, an unofficial Rust client, at 17.0% of (Table 4). The remaining 31 clients make up 6.4% of all peers. The third most common client at 5.2% of the network is an unofficial JavaScript client, which could provide an avenue to cryptojacking web clients to mine Ether.

To understand the dynamics of client updates and adoption in Ethereum, we recorded the client versions for Geth and Parity. We find that Geth nodes primarily operated stable releases (81.9% of Geth nodes), but only 56.2% of Parity nodes were stable (Table 5). The most up-to-date stable releases, namely Geth v1.8.12 and Parity v1.10.9, constitute only 0.6% of Geth nodes and 0.1% of Parity

Client	Count	Percentage
Geth	132,554	76.58%
Parity	29,489	17.04%
ethereumjs-devp2p	8,919	5.15%
teth	782	0.45%
Ethereum(J)	659	0.38%
28 others	693	0.40%

Table 4: Ethereum Mainnet Clients—Geth accounts for nearly 3 out of every 4 nodes detected. Of the top 3 clients, which represent 98.77% of all nodes, only Geth is an official implementation.

Geth Ver.	Count (%)	Parity Ver.	Count (%)
v1.8.2-S	31,244 (23.57%)	v1.10.6-S	6,632 (22.49%)
v1.8.11-S	13,750 (10.37%)	v1.10.4-S	1,698 (5.76%)
v1.8.10-S	13,618 (10.27%)	v1.11.1-U	1,471 (4.99%)
v1.8.8-S	9,573 (7.22%)	v1.10.1-U	1,431 (4.85%)
v1.8.7-S	7,236 (5.46%)	v1.11.3-U	1,399 (4.74%)
v1.7.3-S	6,373 (4.81%)	v1.9.7-S	1,293 (4.38%)
v1.7.2-S	6,037 (4.55%)	v1.9.5-S	1,177 (3.99%)
v1.8.6-S	4,934 (3.72%)	v1.10.3-S	1,099 (3.73%)
v1.8.3-S	4,142 (3.12%)	v1.10.2-U	1,093 (3.71%)
v1.8.4-S	3,449 (2.60%)	v1.7.8-S	878 (2.98%)
101 others	32,198 (24.29%)	99 others	11,318 (38.38%)
Total	**132,554 (100%)**	**Total**	**29,489 (100%)**

Table 5: Geth and Parity Versions—The top 10 Geth and Parity versions' build types are labeled as (S)table or (U)nstable.

nodes. This is not surprising as they were released on July 5th and 7th respectively and our data collection stopped on July 8th. The difference between the two clients' deployment approach becomes more clear as we look at older versions. Among the Geth's top 10 versions, we find the 8 most recent stable versions (excluding v1.8.5 and v1.8.9 which were quickly replaced with next iterations to fix deadlock issues [52]). The positive correlation between a version's freshness and its popularity indicates steady and uniform deployments of updates. This is achieved by Geth's simple deployment cycle: as current unstable version's status updates to `stable`, next version number is given the `unstable` status and its development begins. In comparison, we find both stable and unstable releases and various versions among the Parity population's top 10 versions. This is because newer versions of Parity are released at various states (e.g., stable, beta, and release candidate) every week [48]. The fast-paced development cycle also explains why Parity's version distribution is more sparse than Geth's.

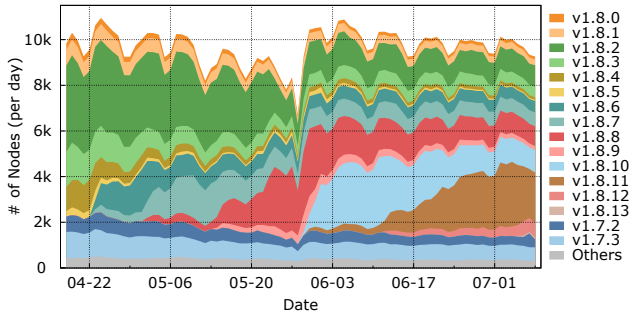

Figure 10: Geth Version Distribution over Time—Upon a new version release, number of nodes running the new version sharply increases as the previous version's population starts declining.

While we observe majority of Geth and Parity nodes updating to new versions, Figure 10 shows an insignificant number of Geth nodes continuing to run old versions. On July 8th (the last date in our dataset), 68.3% were running versions older than 2 iterations (v1.8.10 or lower). The population of v1.7.2 and v1.7.3 has been slowly decreasing but almost 1K nodes were still running them. Although newer versions do not necessarily provide better compatibility, security, or performance, some versions, like v1.7.1 the first version fully compatible with Byzantium hardfork [20], implement substantial changes required to remain compatible with the network or to fix critical bugs. We find 3.5% of the Geth nodes running versions older than v1.7.1, most likely unable to move past the hardfork block.

6.3 Geth/Parity Friction

While examining RLPx connections between Parity and Geth nodes, we discovered an incongruity between Parity's XOR metric and Geth XOR metric. Specifically, Geth implements the log distance metric correctly by calculating the log distance on the XOR of two node IDs' Keccak-256 hashes. Parity, on the other hand, calculates the log distance on each byte of the XOR value and sums them

(see Appendix A). These differing log distance calculations for Geth (ld_G) and Parity (ld_P) have the following relationship:

$$ld_P(x,y) = ld_G(x,y) \iff y = 2^{ld_G(x,0)} - 1 \quad (1)$$

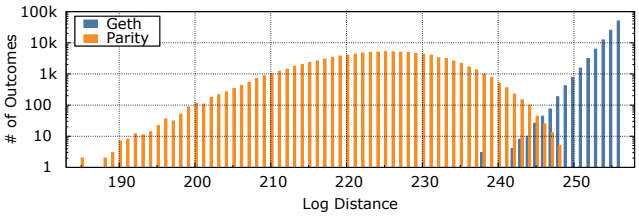

Figure 11: Geth/Parity Node Distance Dist.—We simulated 100K random distance calculations for Geth and Parity. Parity calculates node distance improperly and generates a vastly different range and binomial distribution, while Geth demonstrates an exponential distribution.

As shown in Figure 11, we simulated the node distance distribution for both Geth and Parity (100K trials each) to demonstrate the disparity between the two client types. This differing behavior does not break Geth and Parity compatibility, but it likely hampers the performance of RLPx node discovery. In the best case, Parity peers are effectively useless during Geth's recursive `FIND_NODE` process for converging on nodes closest to a randomly generate node ID. In the worst case, a Geth node with a Parity-saturated RLPx table could fail to discover new nodes since no `NEIGHBORS` response messages would contain new, close nodes for Geth to iterate on. This is effectively an unintentional eclipse attack that could arise naturally on the Ethereum network. Because our measurement does not infer peer topology, we cannot verify whether this inadvertent attack occurs in the wild.

7 P2P COMPARISON

7.1 Network Size

In order to make a fair comparison between our results and the previously reported sizes of Ethereum, Bitcoin, and Gnutella's P2P network, we counted the number of nodes seen over a 24 hour period on April 23rd. This is the same period we considered for our external validation (Section 5.3), and we exclude abusive nodes (Section 5.4) from the dataset. Table 6 summarizes our findings. Compared to other Ethereum measurements [22, 26], NodeFinder is able to find 2.3 times more nodes. On the other hand, Ethereum's 15,454 nodes is significantly smaller than the 62,586 node Gnutella network as measured by Leskovec and Krevl of Stanford's SNAP [37] in August 2002. Both Gencer et al. [26] and Bitnodes measured publicly-reachable nodes only and reported 4,302 and 10,454 respectively.

Network	Date	Size
Ethereum (NodeFinder)	04/23/2018	15,454
Ethereum (Ethernodes [22])	04/23/2018	4,717
Ethereum (Gencer et al. [26])	–	4,302
Bitcoin (Bitnodes [1])	04/23/2018	10,454
Gnutella (SNAP [37])	08/31/2002	62,586

Table 6: P2P Network Size—NodeFinder observes 2.3× more Ethereum nodes than prior methods. Bitnodes and Gencer et al. only connect to publicly-reachable nodes, while NodeFinder and Ethernodes' measure incoming, publicly unreachable nodes as well.

While the 24-hour period snapshot provides a view sufficient for comparing with other P2P networks, it does not provide a good representation of the network's instantaneous state as it is influenced by both IP churn and node churn. To better estimate the state of the Mainnet, we look at 8,309 nodes seen during a one hour period between between 1pm and 2pm on May 8th of 2018 (UTC). We observed 6,599 nodes that support the DAO fork and 518 Classic nodes that oppose the fork. In the following sections, we look at distribution of the 6,760 non-Classic Ethereum Mainnet nodes from this 1-hour snapshot, including 161 nodes that have not yet reached the DAO fork block and can potentially become peers on either network. This dataset is used for subsequent snapshot analysis.

7.2 Geography and Network Distribution

Ethereum connectivity is determined by random node ID lookups, which are independent of geographic location or network address. However, understanding Ethereum's geographic and network distribution can still inform which countries and autonomous systems (ASes) have the largest potential impact on Ethereum. In Figure 12, we find 43.2% of the Mainnet nodes operating in the US, 12.9% in China. From an AS perspective, we see that the top 8 ASes account for 44.8% of nodes and are all cloud hosting providers including Amazon, Alibaba, Digital Ocean, OVH, Hetzner, and Google. This suggests that Ethereum nodes operate primarily in cloud environments, rather than residential or commercial networks.

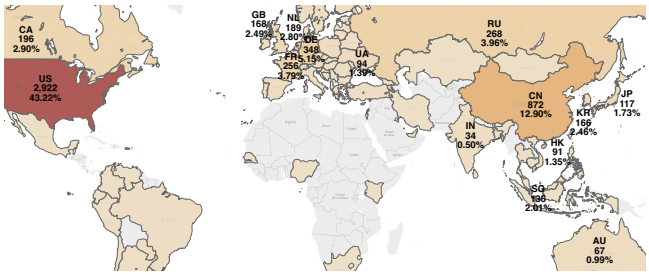

Figure 12: Geographic Distribution—The top 3 countries by number of Ethereum nodes are US (43.2%), China (12.9%), and Germany (5.2%). Ethereum is a random network by design, so geographic location should not affect network connectivity.

In a previous study on the Gnutella network, Saroiu et al. [53] measured latencies between their measurement node and Gnutella peers to verify that the network is formed in an unstructured (ad-hoc) way. To compare Ethereum P2P network's latencies to those of other P2P networks, we first calculated latencies based on

Pct %	Latency (ms)				
	Ethereum		Gnutella [53]	Bitcoin [26]	Ethereum [26]
	Dir	Tri	Dir	Tri	Tri
10%	25	99	–	48	92
20%	51	116	70	–	–
33%	101	151	–	79	125
50%	132	208	–	109	152
67%	192	231	–	152	200
80%	228	247	280	–	–
90%	250	285	–	286	276
Avg.	302	209	–	135	171
Std. Dev.	267	157	–	88	76

Table 7: P2P Peer Latencies—Our triangle (Tri) inequality estimate yields latency estimates similar to a prior Ethereum study [26], which revealed a higher average latency and std. dev. than Bitcoin. Direct (Dir) connection measurement demonstrates lower latency than Gnutella.

smoothed round-trip times measured on direct connections, which NodeFinder recorded for every peer connection as explained in Section 4. Gencer et al. [26] utilized the triangle inequality technique [24] for estimating lower and upper bounds between two remote nodes. We used the average of the bounds to estimate latency. Table 7 summarizes the latencies for Gnutella, Bitcoin, and Ethereum. Our Ethereum measurements parallel the measurements by Gencer et al. and revealed a higher latency and broader distribution than Bitcoin, which indicates that Ethereum is likely formed in a random, ad hoc manner.

7.3 Client Age

In Figure 13, we tracked how often Ethereum node operators update their clients by manually labeling Geth and Parity versions with their release dates. Nearly 50% of Parity clients and 75% of Geth clients are less than three months old. This resembles the BitTorrent network in 2007, when 50% of BitTorrent clients were found to be than 3 months old [12]. Because Geth has been in development for a longer period of time, we find a larger proportion of old (> 4 months) Geth clients than Parity clients. On the flip side, we see that active Geth updaters are more up-to-date than Parity clients, despite the fact that Parity supports configurable automatic upgrades, and Geth does not. As noted in Section 6.2, delayed updates can lead to vulnerable nodes that may even become incompatible with other nodes on the network in the event of a hardfork or other breaking changes.

7.4 Node Freshness

We evaluate node freshness based on how close the peers were to the head of the blockchain during the analyzed time frame. Figure 14 shows the CDF of node freshness based on block numbers determined from the `bestHash` field of the peers' `STATUS` messages. We observe that 32.7% of the nodes were stale and did not actively contribute to the Ethereum network as they couldn't have validated and propagated transactions with unsynced blockchains. Additionally, 141 nodes were found to be stuck at block 4,370,001—the first

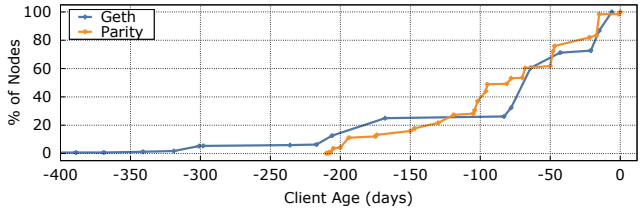

Figure 13: Ethereum Client Age—Approximately 50% of Parity clients and 75% of Geth clients are less than 100 days old.

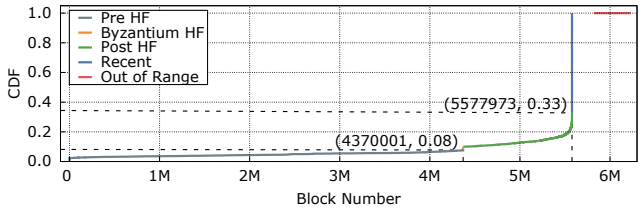

Figure 14: Node Freshness—67.3% of nodes are up-to-date with the Ethereum blockchain. 2% are stuck at the Byzantium hardfork block.

block after the Byzantium hardfork—most likely due to running versions incompatible with the hardfork as discussed in Section 6.2. The stale one-third may be closely related to the network's churn rate or incorrect implementations of the node discovery algorithm resulting in failure to maintain uniformly random network topology uniform and low diameter. This remains an area for future work.

8 DISCUSSION

Our deployment of NodeFinder shed light on the size, decentralization, performance, and behavior of the Ethereum P2P network from both a longitudinal ecosystem view, as well as a single snapshot. We found evidence of multiple network inefficiencies and other areas of concern including outdated clients susceptible to patched vulnerabilities and hardfork incompatibility. These discoveries point to a rapidly changing, immature network that has yet to effectively adopt some of the best practices established by other network systems. These include:

Automatic updates. Although Ethereum exhibits client update rates similar to previously studied file-sharing P2P networks, Ethereum clients would benefit greatly from aggressive automatic updates since vulnerabilities can and have resulted in major financial consequences [25, 49]. Coincidentally, Ethereum clients are inherently always online and good candidates for updates.

Improved standardization. RLPx, DEVp2p, and Ethereum subprotocol are generally viewed as being insufficiently documented [26, 40], which reflects our experience while investigating them. Poor standardization of Ethereum's network protocols is a likely contributor to conflicting client implementations such as Parity's miscalculated XOR metric. Comprehensive documentation and a reference

test specification would reduce the incongruences between different clients and improve overall network efficiency.

Active monitoring. More broadly, our characterization of the Ethereum P2P network highlights the importance of active monitoring as a tool for improving network robustness via measurements of its performance, availability, and security. Systems similar to NodeFinder in other domains have been useful for identifying network issues, notifying affected parties, and ultimately deploying mitigations [1, 17]. NodeFinder is an open-source foundation for future Ethereum studies and monitoring efforts. We plan to extend Ethereum monitoring with new ethical measurement techniques to further understand Ethereum's peer topology and its relation to Ethereum's application layer.

9 RELATED WORK

The main value of Ethereum (and cryptocurrencies in general) is to provide distributed, decentralized consensus on valid transactions and smart contracts. Formal modeling has demonstrated the fault tolerance of Bitcoin [42] when Byzantine faults account for less than half of the network, with similar results applying to Ethereum [7]. Several replacement Byzantine consensus protocols have been proposed to improve scalability and reduce consensus latency [23, 34].

In practice, the performance and robustness of Byzantine consensus protocols are highly reliant on network connectivity, which can cause a small number of *failures* to translate into a large number of Byzantine *faults*. For instance, certain network topologies can be fully partitioned by a few network failures, leading to large subgraphs that are considered Byzantine faults. Network topology is especially important in random graph networks, and many algorithms have been proposed for distributed consensus in sensor networks [32, 33, 45]. Ethereum also operates on a random graph by design, but no work has examined the properties of the Ethereum network, leaving the possibility of brittle network and high consensus delay. Initial evaluation of information propagation in the Bitcoin network has demonstrated that slow consensus can lead to high occurrence of forks, which further delays consensus and increases wasted computation [13].

Insight into Bitcoin's P2P network has previously exposed a network-level vulnerability that enabled application level attacks [31]. Specifically, an attacker could monopolize all peer connections on a victim node and effectively launch selfish mining, adversarial forks, and double spending attacks. To our knowledge, only Gencer et al. have previously examined Ethereum's network properties to quantify the degree of decentralization. By measuring Ethereum's provisioned bandwidth and pairwise peer latency, the authors observed that Ethereum nodes are more widely distributed than Bitcoin nodes but have less spare bandwidth. The study does not explore the RLPx or DEVp2p layers of Ethereum's P2P network; further, they provide no indication that they focus on Ethereum's network to the non-Classic Mainnet nodes which underlie the cryptocurrency. From an adversarial perspective, two very different eclipse attacks have been demonstrated. The first attack poisoned the blockchain synchronization process to continuously feed malicious blocks to the victim, and prevent it from advancing its blockchain [57]. The second set of attacks, introduced by Marcus et al. targets the RLPx peer establishment process by overwhelming monopolizing a node's peer table, which is flushed immediately after reboot [40]. They also introduce an orthogonal attack that relies on compromising a node's NTP service. This paper complements these studies by uncovering the applicability and potential impact of such attacks, in addition to identifying other network properties or incongruences relating to availability or security.

Outside of the cryptocurrency domain, there has been much effort on studying properties of P2P networks, particularly in file sharing systems. Saroiu et al. crawled the Napster and Gnutella networks and identified their properties, such as population, bottleneck bandwidths, latencies, availability, churn, and degree of connections [53]. Qiao et al. performed a combination of passive and active measurement on Gnutella and Overnet networks and evaluated their resilience, message overhead, and query performance [51]. Pouwelse et al. studied the BitTorrent network over a period of 8 months and found that the network suffers from the unavailability of the network's central components, such as seed trackers. Dinger et al. analyzed performance of two decentralized bootstrapping approaches—namely local host caches and random address probing—in the BitTorrent network [16]. We perform direct comparison of these networks with Ethereum's P2P network throughout the study.

10 CONCLUSION

Due to their ballooning financial value, the robustness (i.e., performance, availability, and security) of cryptocurrencies has recently come under high scrutiny. Recent work investigating the peer-to-peer network underlying Bitcoin has demonstrated that network robustness is imperative for proper blockchain operation. In this work, we developed NodeFinder, a novel monitoring tool that provides an unprecedented view of Ethereum's network ecosystem. We uncovered a cluttered network that contains thousands of nodes running various non-Ethereum services. Compared to other P2P networks, Ethereum exhibits similar network performance properties, but also shows hallmark signs of a small, immature P2P network that would benefit greatly from adoption of known best practices and additional examination of the concerns presented in the paper.

ACKNOWLEDGMENTS

The authors thank their shepherd Fabián E. Bustamante and the anonymous reviewers for providing invaluable comments and suggestions to improve the quality of the paper.

This work was supported in part by the National Science Foundation under contract CNS 1518741, by the Department of Homeland Security under contract HSHQDC-17-J-00170, and by a gift from Google. This work was additionally supported by gifts from Jump Labs and CME Group. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

REFERENCES

[1] 21.co. 2018. Global Bitcoin Nodes Distribution. Retrieved April 23, 2018 from https://bitnodes.21.co

[2] ArabianChain Technology. 2017. Retrieved May 18, 2018 from https://www.arabianchain.org/
[3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on Ethereum smart contracts (SoK). In *International Conference on Principles of Security and Trust*.
[4] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. 2012. The Menlo Report. *IEEE Security and Privacy* (2012).
[5] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *Financial Cryptography and Data Security*.
[6] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2011. The Keccak SHA-3 submission. *Submission to NIST (Round 3)* (2011).
[7] Vitalik Buterin. 2016. Proof of Stake FAQ. Retrieved October 26, 2017 from https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
[8] Vitalki Buterin. 2017. Ethereum White Paper. Retrieved November 29, 2017 from https://github.com/ethereum/wiki/wiki/White-Paper
[9] Vitalik Buterin. 2017. Ethereum Wire Protocol. Retrieved November 12, 2017 from https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol
[10] Vitalik Buterin. 2018. Light client protocol. Retrieved September 13, 2018 from https://github.com/ethereum/wiki/wiki/Light-client-protocol
[11] CoinMarketCap. 2018. Cryptocurrency Market Capitalizations. Retrieved September 12, 2018 from https://coinmarketcap.com/
[12] Scott A Crosby and Dan S Wallach. 2007. *An analysis of Bittorrent's two Kademlia-based DHTs*. Technical Report.
[13] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE P2P Proceedings*. IEEE, 1–10.
[14] Michael del Castillo. 2016. Ethereum Executes Blockchain Hard Fork to Return DAO Funds. Retrieved May 8, 2018 from https://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds/
[15] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. 2016. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *Financial Cryptography and Data Security*.
[16] Jochen Dinger and Oliver P Waldhorst. 2009. Decentralized bootstrapping of P2P systems: A practical view. In *International Conference on Research in Networking*. Springer, 703–715.
[17] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *ACM CCS*.
[18] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications.. In *USENIX Security*.
[19] Ethereum Foundation. 2017. What is Ether. Retrieved November 12, 2017 from https://www.ethereum.org/ether
[20] Ethereum Team. 2017. Byzantium HF Announcement. Retrieved May 23, 2018 from https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/
[21] EthereumJS. 2018. Releases. Retrieved May 21, 2018 from https://github.com/ethereumjs/ethereumjs-devp2p/releases
[22] ethernodes.org. 2017. The ethereum node explorer. Retrieved September 12, 2018 from https://www.ethernodes.org/network/1
[23] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI*.
[24] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. 2001. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions On Networking* (2001).
[25] Sean Gallagher. 2017. With deletion of one wallet, $280M in Ethereum wallets gets frozen. Retrieved May 25, 2018 from https://arstechnica.com/information-technology/2017/11/with-deletion-of-one-wallet-280-m-in-ethereum-wallets-gets-frozen/
[26] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. 2018. Decentralization in Bitcoin and Ethereum Networks. (2018).
[27] Taylor Gerring. 2014. Building the decentralized web 3.0. Retrieved November 12, 2017 from https://blog.ethereum.org/2014/08/18/building-decentralized-web/
[28] Taylor Gerring. 2017. What is Ethereum. Retrieved November 12, 2017 from https://github.com/ethereum/wiki/wiki/What-is-Ethereum
[29] go-ethereum. 2018. Go Ethereum. Retrieved September 12, 2018 from https://github.com/ethereum/go-ethereum
[30] go-ethereum. 2018. Release Bugfoot (1.2.1). Retrieved May 8, 2018 from https://github.com/ethereum/go-ethereum/releases/tag/1.2.1
[31] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *USENIX Security*.
[32] S. Kar and J. M. F. Moura. 2008. Sensor Networks With Random Links: Topology Design for Distributed Consensus. *IEEE Transactions on Signal Processing* (2008).
[33] S. Kar and J. M. F. Moura. 2009. Distributed Consensus Algorithms in Sensor Networks With Imperfect Communication: Link Failures and Channel Noise. *IEEE Transactions on Signal Processing* (2009).
[34] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *USENIX Security*.
[35] Felix Lange. 2017. DEVp2p Wire Protocol. Retrieved November 12, 2017 from https://github.com/ethereum/wiki/wiki/%C3%90%CE%9EVp2p-Wire-Protocol
[36] Felix Lange. 2018. Node Discovery Protocol v4. Retrieved September 19, 2018 from https://github.com/ethereum/devp2p/blob/master/discv4.md
[37] Jure Leskovec and Andrej Krevl. 2014. SNAP: Network datasets: Gnutella peer-to-peer network, August 31 2002. Retrieved May 24, 2018 from https://snap.stanford.edu/data/p2p-Gnutella31.html
[38] Alex Leverington. 2017. RLPx: Cryptographic Network & Transport Protocol. Retrieved November 12, 2017 from https://github.com/ethereum/devp2p/blob/master/rlpx.md
[39] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *ACM CCS*.
[40] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. Cryptology ePrint Archive, Report 2018/236. https://eprint.iacr.org/2018/236.
[41] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the XOR metric. In *International Workshop on Peer-to-Peer Systems*.
[42] Andrew Miller and Joseph J LaViola Jr. 2014. Anonymous Byzantine consensus from moderately-hard puzzles: A model for bitcoin. https://nakamotoinstitute.org/static/docs/anonymous-byzantine-consensus.pdf. (2014).
[43] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2015. Discovering Bitcoin's public topology and influential nodes. (2015).
[44] MOAC. 2018. Mother Of All Coins. Retrieved May 18, 2018 from https://moac.io/
[45] L. Moreau. 2004. Stability of continuous-time distributed consensus algorithms. In *IEEE CDC*.
[46] T. Neudecker, P. Andelfinger, and H. Hartenstein. 2016. Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network. In *IEEE UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*.
[47] Parity Technologies. 2018. Parity - fast, light, and robust Ethereum client. Retrieved May 25, 2018 from https://github.com/paritytech/parity
[48] Parity Technologies. 2018. Releases. Retrieved September 13, 2018 from https://github.com/paritytech/parity-ethereum/releases
[49] Nathaniel Popper. 2016. A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency. Retrieved May 25, 2018 from https://www.nytimes.com/2016/06/18/business/dealbook/hacker-may-have-removed-more-than-50-million-from-experimental-cybercurrency-project.html
[50] P.P. 2016. Top answer to "How many nodes are there on the Ethereum network?". Retrieved September 12, 2018 from https://ethereum.stackexchange.com/a/199
[51] Y Qiao and FE Bustamante. 2006. Structured and unstructured overlays under the microscope. In *USENIX Security*.
[52] James Ray. 2018. Releases. Retrieved May 8, 2018 from https://github.com/ethereum/wiki/wiki/Releases
[53] Stefan Saroiu, P Krishna Gummadi, Steven D Gribble, et al. 2002. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, Vol. 2002. 152.
[54] Péter Szilágyi. 2018. eth/63 fast synchronization algorithm. Retrieved May 15, 2018 from https://github.com/ethereum/go-ethereum/pull/1889
[55] Gavin Wood. 2017. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Retrieved November 29, 2017 from https://gavwood.com/paper.pdf
[56] Gavin Wood. 2018. libp2p Whitepaper. Retrieved September 19, 2018 from https://github.com/ethereum/wiki/wiki/libp2p-Whitepaper
[57] Karl Wüst and Arthur Gervais. 2016. *Ethereum Eclipse Attacks*. Technical Report. ETH Zurich.
[58] Yu-Te Lin. 2017. Istanbul Byzantine Fault Tolerance. Retrieved May 18, 2018 from https://github.com/ethereum/EIPs/issues/650
[59] Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Erays: Reverse Engineering Ethereum's Opaque Smart Contracts. In *USENIX Security*.

A PARITY NODE DISTANCE

```
fn distance(a: &H256, b: &H256) -> u32 {
  let d = *a ^ *b;
  let mut ret:u32 = 0;
  // iterate over each byte of 256-bit Keccak hash
  for i in 0..32 {
    let mut v: u8 = d[i];
    // get the position of first non-zero bit
    while v != 0 {
      v >>= 1;
      ret += 1;
    }
  }
  ret
}
```