Under the Hood of the Ethereum Gossip Protocol

Lucianna Kiffer[1], Asad Salman[1], Dave Levin[2], Alan Mislove[1], and Cristina Nita-Rotaru[1]

[1] Northeastern University, Boston MA,USA
[2] University of Maryland College Park MD,USA
`{lkiffer,amislove}@ccs.neu.edu`, `me@asad.co`, `dml@cs.umd.edu`, `c.nitarotaru@northeastern.edu` **

Abstract. Blockchain protocols' primary security goal is consensus: one version of the global ledger that everyone in the network agrees on. Their proofs of security depend on assumptions on how well their peer-to-peer (P2P) overlay networks operate. Yet, surprisingly, little is understood about what factors influence the P2P network properties. In this work, we extensively study the Ethereum P2P network's connectivity and its block propagation mechanism. We gather data on the Ethereum network by running the official Ethereum client, `geth`, modified to run as a "super peer" with many neighbors. We run this client in North America for over seven months, as well as shorter runs with multiple vantages around the world. Our results expose an incredible amount of churn, and a surprisingly small number of peers who are actually useful (that is, who propagate new blocks). We also find that a node's location has a significant impact on when it hears about blocks, and that the precise behavior of this has changed over time (e.g., nodes in the US have become less likely to hear about new blocks first). Finally, we find prune blocks propagate faster than uncles.

1 Introduction

Ethereum [34] is a cryptocurrency that can also store and execute user-generated programs often called *smart contracts*. Compared to Bitcoin [32], Ethereum is significantly more expressive and can be used to implement decentralized voting protocols, financial contracts, and crowdfunding programs. Today, Ethereum is the second-most-valuable cryptocurrency behind Bitcoin, with a market capitalization of over $26B [6].

Given its complexity, it may be unsurprising that Ethereum has a number of differences from Bitcoin. For the purposes of this paper, two such differences stand out: *First*, Ethereum is based on a general purpose peer-to-peer (P2P) overlay responsible for discovering other nodes and maintaining connectivity. This P2P layer can be used by higher-level protocols other than Ethereum (in

** We thank the anonymous reviewers and Arthur Gervais for their helpful comments. This research was supported in part by NSF grants CNS-1816802 and CNS-1900879, a Ripple unrestricted gift, and Facebook Fellowship. We also thank the Ethereum Foundation for a gift of AWS credit used toward the collection of our data.

fact, we find this is often the case). The Ethereum Network layer sits between the P2P layer and the overlying application layer (the blockchain itself), and is responsible for choosing peers, disseminating new blocks, and reaching network-wide confirmation of transactions. *Second*, Ethereum's 15 seconds (on average) block interval is dramatically shorter than Bitcoin's 10 *minutes* (on average). This significantly reduced target block mining interval opens the door to much faster "network confirmation" of accepted transactions.

Thus, the structure of the Ethereum P2P overlay, including aspects such as peer connectivity and block propagation delay, play an important role in achieving the target performance and correct functioning. Most prior measurement work on Ethereum has focused either primarily on the P2P layer [15, 28] or primarily on the application layer (i.e., the blockchain itself [27, 26, 8, 9, 33]), leaving the Ethereum network layer not as well understood.

In this paper, we aim to better understand the network structure of Ethereum, focusing on both how the Ethereum network is formed and evolves over time, as well as how the network is used to propagate new blocks, a crucial part of the consensus mechanism. We do this by integrating information from the P2P layer (e.g., which nodes are available, who nodes choose to connect to), information from the application layer (e.g., which blocks are ultimately accepted), along with information from the Ethereum layer (e.g., which peers nodes exchange block information with, and which peers actually provide the most useful information).

We conduct our study by running a customized version of Ethereum's official Go client, `geth`, for over seven months, allowing us to observe the evolution of the Ethereum network through multiple protocol changes. We also run multiple nodes in a variety of vantage points across the globe for shorter lengths during this time period, allowing us to study both how our peers interact and how the location in the physical Internet affects the peers' experience in the Ethereum network. The results of our analysis can be summarized as follows:

- **Extensive peer churn:** We observe dramatic levels of churn in the Ethereum network, both in terms of the number of unique peers and connection lengths. Churn can run the risk of disconnecting a network or making it difficult to quickly propagate information throughout it, and challenging to estimate the size of the network. We investigate churn in the Ethereum network and find that 68% of the peer IDs we see in our 200-day scraping period are present only on a single day, and 90% of them are present on fewer than 25 days.
- **Miner centralization:** The top 15 miners are responsible for over 90% of mined blocks. We investigate those miners and find that all but one are well-known mining pools. We also note a difference in the efficiency of miners: the top 3 mining pools have a much greater probability of a block they mine being included in the blockchain (propagating faster and "winning" the block race).
- **Most announcers are long-lived:** Comparing all peers to those who are first to announce a block, we see that those announcing blocks tend to have longer connections, larger average and total connection lengths, and are online more days. Almost 70% of peers are seen only one day, but about 40% of our

announcing peers are seen only one day. The latter is still a large percent but there is a set churn in peers who are not participating in block propagation.

- **Announcers are diverse:** Focusing on *which* peers tell our nodes about new blocks first, we find that a large number of peers are responsible for announcing a miner's block first to our node. No one peer announces more than 6% of a miner's blocks to us first.
- **Quick network-wide propagation:** The speed at which new information spreads throughout the Ethereum network is critical in how quickly transactions can complete. We perform a novel analysis of network propagation times by running nodes in three different vantage points (USA, South Korea, and Germany). Despite the diversity of information sources across the three, we find that the difference in time from when the first learns about a block until when the last learns about it is very small: for instance, less than 100 ms for 85% of all new blocks.
- **Location bias:** Although our vantage points in North America, Asia, and Europe did not experience any unfair disadvantages, we observed bias using additional vantages in locations with fewer peers (South America and Oceania). We observe an significant disparity in which locations hear about blocks first, with those two locations being first to hear about blocks only $\sim$3% of the time.

2 Background

The Ethereum system consists of multiple layers. In this section, we provide an overview of the components we study in this paper, and detail related work.

2.1 Overview

The purpose of Ethereum is to create a *blockchain* via proof-of-work *mining*. Unlike systems that function almost exclusively as a currency, Ethereum is made up of transactions that contain either (a) direct transfers of `ETHER` (the currency unit of Ethereum) or (b) transactions that create or call *smart contracts*. Transactions of the second type must also pay to run the computation via `GAS` in extra `ETHER` sent with the transaction. While much work has examined smart contracts (e.g., [27, 9, 33]), we are focused on the Ethereum network itself.

Miners are participants in the Ethereum network that listen on the network for transactions and package them into blocks. To successfully create a block, a miner must verify that all transactions included are valid (including code execution and signatures), include the hash from the most recent block in the miner's chain as well as a Proof-of-Work (proof enough computational "work" was done to create this block). Miners do this "work" by creating blocks whose hash[3] has a minimum specified number of leading zeros. The target number of leading zeros is known as the *difficulty* and is adjusted at every block so that a block will be generated roughly every 15 seconds. The miner whose block becomes part of the chain is

[3] The Ethereum protocol uses their own memory-hard hash function, Ethhash [1].

rewarded with 5 `ETHER` and the `GAS` of the transactions. Because of the high variance in winning a block, miners often come together to form *mining pools* where block rewards are split among the participants. We observe in (Figure 1) that the the top 15 miners (14 of which are known mining pools) mined over 90% of all blocks.

Mainchain, Uncles, and Prunes are different types of blocks in Ethereum. Blocks that are part of the blockchain containing the history of Ethereum are called `mainchain` blocks. However, not all valid blocks share this fate. With a target time between blocks of 15 seconds, it often happens that two or more valid blocks are mined within a short interval of each other by two different miners, contending for the same position in the chain. Both blocks propagate through the network and eventually consensus is reached on which of the two blocks become part of the `mainchain`.

To still reward mining related to these discarded blocks, miners can also choose to include these valid, but non-`mainchain` blocks as `uncles` in the blocks they mine. A miner includes the hash of the discarded block in a special field of the block only if the parent of the `uncle` (the block the `uncle` points to) is a block in their own chain up to six blocks prior. Both the miner and the miner of the `uncle` receive an additional, smaller amount of `ETHER`.

Finally, some valid blocks may be mined, but never end up in the `mainchain` or become `uncles` (e.g. if the announcement of the block is significantly delayed), we refer to such blocks as `prunes`. Note that because `prunes` are not on the blockchain at all, we can only observe them by participating in the network and hearing them being announced. On average, we observe roughly 6,000 `mainchain` blocks, 400 `uncle` blocks, and 10 `prune` blocks each day.

2.2 Networking in Ethereum

The Ethereum system is made up of three layers: the *application layer* that contains the blockchain, the *Ethereum layer* that contains peers exchanging information about blocks and transactions, and the *peer-to-peer (P2P) layer* that allows nodes to find others and establish connections (more details are in Appendix A). We briefly overview these below based on the official documentation [4, 5, 2], talks [20], and the official client code.

P2P layer The P2P layer is divided into two components: a *discovery protocol* that allows nodes to find each other, and *DevP2P* that nodes use to communicate. We detail the Discovery protocol in Appendix A.

The DevP2P protocol runs in parallel to the Discovery protocol and is responsible for establishing sessions with other nodes, sending and receiving messages between peers and managing the actual higher-level protocol being run. In Ethereum, DevP2P uses RLPx, which is responsible for encrypting, encoding and transferring messages between peers. Once a peer has been discovered during the execution of the Discovery protocol, the RLPx protocol initiates the TCP handshake and HELLO messages are exchanged. In the HELLO message, both sides send the protocol version, the client software type, the capabilities and version they support, the port they are listening on and their `ENODEID`.

Importantly, the DevP2P protocol checks if the remote node is running the same application-layer protocol (e.g. `eth` for the Ethereum Wire Protocol), that they support each others' protocol version, and that they agree on the blockchain (i.e., the genesis blocks and any forks). The nodes will disconnect if any of these conditions do not hold, which is surprisingly common: prior work [28] found that over two months in 2018, about 95% of nodes were running the `eth` protocol, but only 54.5% of those agreed on the blockchain. Otherwise, if the conditions hold, the nodes become *peers* and can exchange messages at the Ethereum layer (we distinguish *nodes* at the P2P layer from *peers* at the Ethereum layer).

Ethereum layer The Ethereum Wire Protocol [2] is the application-layer protocol for propagating transactions and blocks, and for requesting block and state data so new clients can sync to the existing state. To be brief, we omit how new clients sync to the blockchain and instead focus on how new messages are propagated.

Transactions are transmitted in full via a `TransactionMsg` message either by the originator of the transaction or when a node hears about a new transaction.

Blocks are transmitted in a more complicated fashion. When a node hears about a new block, they first verify that the block belongs to their chain and includes the PoW. At this point, the node *propagates* the full block by sending a `NewBlockMsg` message to a subset of their peers.[4] The node then fully validates the whole block by adding it to their internal state. The node finally sends the hash of the block to its remaining peers who have not heard about the block via a `NewBlockHashesMsg` message. An overview of this process is provided in Algorithms 1-4 in Appendix B.

2.3 Ethereum implementations

There are several versions of the Ethereum client, the most common are the official Golang implementation called `geth` and a popular, non-official Rust-based client, Parity. We deploy our measurements using the official `geth` client, run by a vast majority of the network [28, 7]. By default, the `geth` client used to have the *maxPeer*, a cap of the number of peers the client will maintain, constant set to 50. This cap is enforced by bounding 66% of its *maxPeer* as incoming connections and the remaining connections as outbound. In order to gain visibility into the network, we increased the *maxPeer* cap to 750 during most of our experiments. The `geth` client will keep on accepting and making connections until its peer cap is met, including looking for additional nodes to connect to via the node discovery protocol.

[4] In `geth` this subset is composed of a square root of their peers who have not heard about the block.

2.4 Related work

The papers that most closely relate to our work are [15] and [28]. In [15], Gencer et al. run a measurement-based comparative study of the Bitcoin and Ethereum P2P networks with a focus on decentralization properties. For Ethereum they look at peer bandwidth, connection latency (to peers and bounds between nodes), and some amount of efficiency of miners through miner distribution of blocks and uncle counts. In [28], Kim et al. scrape the Ethereum P2P network by connecting to peers just long enough to establish a full DevP2P connection and checking up to the DAO fork (i.e. not a ETC node). They focus their analysis primarily on node client type, "freshness", location/ASes and also connection latency. These two papers ran scrapers collecting quick peer information while we run a long-term full node which is able to collect more temporal node information (i.e. analyze churn in more detail), and connect peer information with the kinds of block data they send us (i.e. block propagation, some miner analysis), as well as capture `prune` blocks which has yet to be observed in Ethereum. We can also distinguish exactly the peers who fully participate in the Ethereum protocol as those who send useful block information, i.e. propagate blocks.

We use ethernodes.org to compare the nodes we see and note that there has been work showing how ethernodes.org data is not representative, e.g. many of the peers it reports are not actually running the mainnet Ethereum protocol [28, 3]. They also briefly mention churn, but in no detail. We explore churn in greater detail both in the ethernodes.org data and in our own peer data.

In another Ethereum network measurement study [14], Gao et al. scrape the P2P layer for peers who they make TCP connections with, though similarly to ethernodes.org, a TCP connection does not distinguish mainnet nodes. They enumerate peer tables for those nodes and analyze their topological properties, though peer tables do not represent actual peer connections on the network. Other measurement works in this area include Decker et al. [11], who measure the block propagation delay and fork-rate of Bitcoin[5], work studying peer churn in Bitcoin and other non-blockchain P2P networks [25, 12, 30], and many works analyzing data extract-able from the blockchain [27, 26, 8, 9, 33].

3 Methodology

We now detail our data collection methodology, how we processed the resulting data set, and provide a high-level overview of the data we collected.

3.1 Ethereum client

We created an instrumented and customized version of the `geth` client [18] that was designed to log detailed information about its network- and application-layer activity. At the P2P layer, our client logs all attempted connections (both inbound and outbound, called *handshakes*) along with remote node information

[5] find a median and mean delay of 6.5 and 12.6 seconds (from 2013)

including the remote node's `ENODEID`, IP address, and announced software version. Our client also logs all `PING` and `PONG` messages that Ethereum periodically sends as "heartbeats" between nodes (measuring network latency).

At the Ethereum layer, our client logs a number of messages that are exchanged between peers, primarily focused on messages concerning blocks (`NewBlockMsg` and `NewBlockHashesMsg`). For each message, our client logs a timestamp and the identity of the remote peer.

To limit any negative impact on the network, our client largely participates in the network in the same manner as a regular full node (e.g., finding peers, exchanging information, etc). There are two primary modifications we make to enable us to understand the Ethereum network: *First*, we modify our client to *suppress announcing and forwarding one-third of* blocks and transactions (blocks whose hash value is a multiple of three). We do so in order to study how those messages are propagated without our client affecting their dissemination; our client *forwards the other two-thirds* of blocks and transactions as normal.[6] *Second*, we modify our client to allow a much higher *peer cap* (the limit on the number of network-layer peers the client will connect to). We do so in order to study the behavior of many remote peers at once and, as Gencer et al. showed in Bitcoin [17], the more connections we maintain, the earlier we receive block information, meaning we are likely *closer* to their sources.

3.2 Data collection

We conducted three runs of data collection with different numbers and locations of our clients. We describe these below. In all cases, we use Amazon Web Services' EC2 to host our client, using a `r5.2xlarge` machine time to ensure the hardware had sufficient capacity. We configured the host operating system to sync with timeservers via the Network Time Protocol service continually to adjust for clock drift. Unless otherwise noted, we set the peer cap in our client to 750 peers (we demonstrate below that we likely connected to the vast majority of other nodes).

We found that the `geth` client (v1.9.0) appeared to have some memory leaks (that were exacerbated by our modification of the peer cap to a much larger level than normal). As a result, our clients would sometimes crash and be immediately restarted. We found that our client would often take a few hours to build up its peer count (e.g., see Figure 5), so for our analysis, we ignore any data from before the client reported at least 400 peers.

Peer Cap Experiment Our choice of a maximum of 750 peers for our long-running measurements was because of a memory leak in the `geth` client. In order to establish whether our choice of 750 peer cap is representative of the network, we ran three clients in parallel in the `us-east-1` Amazon data center (in Virginia, U.S.) with increasing peer limits with peer caps of 500, 1000 and 1,500 peers respectively. We explore this experiment in detail in Appendix C,

[6] We are unable to avoid forwarding information on *all* blocks/transactions, as doing so would cause other peers to decide to stop peering with our client.

		Peer counts					Block counts		
runs	location	udp	p2p	peer	useful	first	main	uncle	prune
longitudinal	**U.S.** ID	1,301,568	194,608	90,265	24,945	12,593			
May-Dec	**U.S.** IP	339,832	138,107	55,091	22,982	9,359	1,179,883	79,938	2,695
	U.S.	119,892	20,339	8,932	2,822	1,331			
vantage	**Seoul**	106,397	23,059	9,876	4,865	1,459			
June 6-10,	**Frank.**	107,559	24,744	10,876	4,695	1,138			
14-16, 2019	**All**	150,961	31,188	15,644	6,526	2,465	39,345	2,977	113
	U.S.	299,971	21,257	9,234	5,055	1,214			
information	**Seoul**	295,480	21,024	8,631	5,175	1,822			
propagation	**Frank.**	289,902	20,708	8,276	5,071	1,349			
May 12-	**São P.**	290,390	22,112	9,440	5,337	2,152			
23, 2020	**Sydney**	307,173	23,225	9,643	5,556	1,559			
	All	480,355	30,896	14,613	6,913	3,867	69,199	4779	153

Table 1: Peer and block counts for the *longitudinal experiment*, *multiple vantage point experiment* and *information propagation experiment*. For the *longitudinal experiment* we look at unique `ENODEID` and IP, while for the rest we look at `ENODEID`. We note that `useful` refers to any peer who announces blocks to us while `first` refers to peers to who are the first to announce a `mainchain` block to us.

and the results suggest that the reachable peers we can connect to at any given time caps at about 1,000 peers.

Longitudinal experiment Our primary data collection experiment was a long-term longitudinal study of how the Ethereum network behaved over a period of many months. We refer to this experiment as the *longitudinal experiment*, and it consisted of a single client running in the Virginia Amazon data center between May 15th, 2019 and December 13th, 2019.

Multiple vantage point experiment For a shorter period of time, we also ran a client in the `ap-northeast-2` Amazon data center (Seoul, South Korea) as well as a client in the `eu-central-1` Amazon data center (Frankfurt, Germany), alongside our Virginia, U.S. client. We refer to this experiment as the *multiple vantage point experiment*, and was run in 2019 between June 6th and June 10th and then again between June 14th and June 16th.

For this experiment, we only consider times where all three nodes were up (with a sufficient peer list, as described above). In total, this experiment resulted in 6 days and 4 hours of logged messages.

Information propagation experiment In our multiple vantage point experiment, we observed that our three chosen vantage points tended to be physically close to where most of the blocks were first being announced from(Table 3) and where the majority of our peers are located(Table 2). We finally ran one additional experiment with our three locations in the multiple vantage point experiment, as well as a node in the `sa-east-1` Amazon data center (São Paulo, Brazil) and the `ap-southeast-2` Amazon data center (Sydney, Australia). We chose these two additional locations as they appeared to be locations where very few (<1%) of blocks are being first announced from, and including them would allow us to better understand how network location affects when information in the Ethereum network is received. We ran this experiment between May 12th and 23rd, 2020, and we refer to it as the *information propagation experiment*.

Limitations We note that since these are all EC2 instances, running copies of the same machine in different Amazon locations allowed us to maintain location as our only variable (so hardware differences would not affect our results) as well as have the storage and memory capabilities needed. A clear limitation is whether Amazon machines have a biased view of the network, including special links between their centers we cannot control for. We note that a quarter of all peers we connect to in the *longitudinal experiment* are running on EC2 instances, the largest fraction from a single provider. We also weighted this choice with the additional variable of using multiple cloud providers or VPNs which would have added artificial latencies to our connections.

Ethernodes Ethernodes [7] is a public web site that reports on aggregate statistics for the Ethereum network and is widely cited when reporting statistics about the Ethereum network, including in academic work [16, 19, 13, 24, 23]. Between March 30 and October 15th, 2019, we scraped Ethernodes for the Ethereum node information they report from their crawler. In Appendix E we use the Ethernodes data as a point of comparison for our data analysis.

4 Analysis

We organize the analysis section by working our way down the different layers involved in the Ethereum protocol. We are interested both in understanding general trends in the network and narrowing in on specifics related to peer behavior. Unless otherwise specified, the bulk of the analysis refers to data from the *longitudinal experiment*.

We start with the **Application layer**, i.e. who is mining blocks. Our main questions are: *How is mining distributed among the top miners, and are the top pools equally efficient? In other words, do some miners appear to have an advantage (e.g., are less likely to mine non-*`mainchain` *blocks)?*

Next, we move on to the **Ethereum layer** by examining the timestamps of when we hear about different types of blocks. Here we are interested in answering: *How are blocks being propagated in Ethereum and is block propagation correlated to the type of block, block size, or other factors?*

We then examine the **P2P layer**, where we focus on who our nodes connect to/come across. The bulk of the novelty in our work comes from tying peer connectivity behavior with its *usefulness* in block propagation. This is done in the following two layers both in breaking down the behavior of our peers in the *longitudinal experiment* and comparing information received from peers from different vantages. We ask: *What trends can we observe in peer connectivity behavior? How many of the peers that we come across end up being* `useful`*?*

Finally, we end our analysis by looking at the **underlying network** and how the position of our nodes in the Internet affects their view of the network. For this we utilize both the *multiple vantage point experiment* and the *information* *propagation experiment* to answer: *Are there advantageous geographic locations from which to run an Ethereum peer and, if so, how large is the disparity between locations?*

4.1 Application Layer

We begin by examining who is mining blocks by looking at the self-advertised `miner id` in the blocks our client hears about. We take data from the seven months of the *longitudinal experiment*, and group blocks by whether they are part of the `mainchain`, `uncles`, or `prunes`.

Figure 1 plots the cumulative distribution of the blocks of different types across miners (note the log scale on the x-axis). We can immediately observe that the fraction of blocks mined is not uniform, and in fact highly skewed towards a very small number of miners. While 90% of all blocks are mined by less than 5 % of miners (and the top three miners mine over half of the blocks), over half of the miners mine just a single block. Unsurprisingly, when we examine *when* these blocks are mined, the few miners who mine the majority of the blocks are active for the length of the measurement period, while others who win less frequently come and go more often.

We note that the low number of `miner ids` is not entirely surprising, as miners often group into mining pools and all mine for the same `miner id`. As a result, much of the discrepancy between how many blocks miners win can be explained by dramatic differences in aggregate mining power across pools.

To further explore the discrepancy across miners, we examine each miner's `uncle` to `mainchain` and `uncle` to `prune` ratios in Figure 1. Recall that `uncles` and `prunes` occur when multiple blocks are mined at once, and eventually one wins. Similar to the analysis of Gencer et al. [15], we say if mining was "fair", we would expect that all miners would typically mine a similar fraction of `uncles` and `prunes` (relative to all blocks they mine)[7]. However, we see that

[7] Gencer et al. [15] compared Bitcoin `prunes` to all blocks mined, but for Ethereum just used `uncle` counts. They found that at the time Bitcoin had a larger standard deviation in mining fairness than Ethereum.

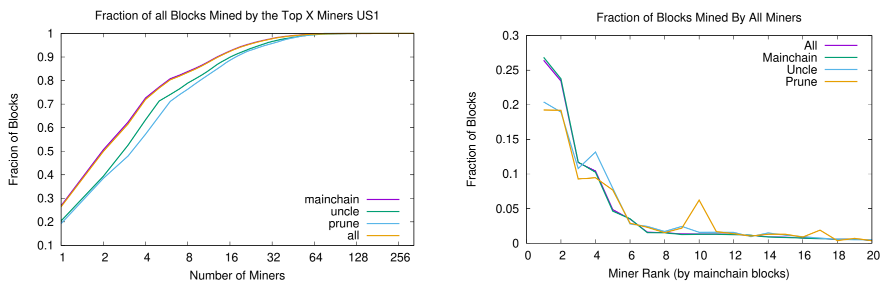


Fig. 1: *Left*: Cumulative distribution of the number of all blocks won by top miners. Note the log scale on the x-axis. *Right*: The fraction of total `mainchain`, `uncle`, and `prune` blocks each miner mines. We see the top 3 miners mine disproportionately more of the `mainchain` blocks than `uncles` or `prunes` and thus have a disproportionate advantage over the other miners.

the top `mainchain` miners tend to have many fewer `uncle` and `prune` blocks than `mainchain` blocks, but that this trend fades as we start to look towards less-powerful miners. This suggests that larger miners appear to have some sort of advantage in the network, as they suffer from `uncles` and `prunes` at a much lower rate. Given a block race, having larger mining power increases a miner's odds of winning (as they are more likely to *win* the next block), and network advantages (lower delays) would further increase their odds. It is unclear at which point the former plays a bigger role.

Next, we examine the behavior of the Ethereum protocol layer to better understand how blocks are propagated in the network.

4.2 Ethereum Protocol Layer

We now explore general block trends and how blocks get propagated in the network. In Figure 2, we examine how our client first hears about blocks (`NewBlockMsg` or `NewBlockHashesMsg`) over time. It is clear that new `mainchain` blocks and `uncles` are primarily announced to our node first as the full block message, though there are times when some blocks get to it first as hashes. This corresponds to our peers first propagating the full block and then the hashes.[8] Starting in late October, we see the `mainchain` count starts decreasing. This was due to the upcoming *difficulty bomb* which causes the difficulty to gradually increase, speeding up as the deadline approaches; this in turn caused blocks to be mined more slowly.[9]

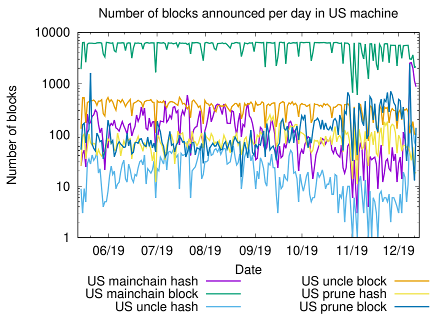


Fig. 2: Count of new blocks by type and by which message we first heard about the new block (NewBlock or NewBlockHash).

We next dig deeper into the `NewBlockMsg` messages to better understand block propagation. Specifically, we look at the incoming `NewBlockMsg` messages for each block, and measure the difference between the *first* time our client hears about a given block and all subsequent times. Figure 3 presents the cumulative distribution of times for different percentiles for each block, broken down by whether or not it was a block that our client propagated (recall we only propagate 2/3 of blocks at random). We can observe that when we propagate blocks, the lower percentiles tend to be *longer* (compared to when we do not). This may be surprising, but is likely due to the fact that other nodes will not inform our client if it knows we already know about a block; by propagating a block, we

[8] When `prunes` are first announced to us via a `NewBlockHashesMsg`, it generally correspond to times we hear about odd blocks that do not follow the `mainchain` (i.e. the block number is much smaller or larger than the current height).

[9] The bomb was delayed with the Muir Glacier hardfork in early January 2020

effectively preclude being told. When we propagate a `mainchain` block we receive fewer announcements for it, which is not true for `prunes` or `uncles`.

We further explore the propagation time for different block types in Figure 3, and note the propagation delay peaks at around 200 ms for all types of blocks. However, `prunes` and `uncles` have a much longer tail, implying their dissemination through the Ethereum network is significantly slower than `mainchain` blocks. We explore whether a block's delay is correlated with any factors about the block. Specifically, we look at the relationship between the median delay of `mainchain` blocks and their `GAS` count and find a very weak correlation coefficient of 0.035. Similarly, we see a similar weak correlation coefficient of 0.031 block size and median delay.

Finally, when we examine the *number* of announcements we receive per block, we notice that we receive between 100 and 300 announcements for most blocks except for `prunes`. The `prune` data is largely skewed by few peers (e.g., 70% of `prunes` are only announced to our client by one peer), and these peers tend to advertise many block hashes with either very low or very high block numbers far from the correct `mainchain`. We note that for these prunes that are both announced primarily as NewBlockHashes (i.e., we do not receive the full block) and in large batches with block numbers that do not correspond to the current height of the `mainchain`, we do not consider them true `prunes` of the `mainchain` and exclude them from the propagation delay analysis of Figure 3.

4.3 Peer-to-Peer Level

We now turn to examine the P2P protocol layer by looking at trends in our connections to nodes. In Figure 4 (left), we plot the number of unique nodes our client `PING`/`PONG`s, starts a TCP handshake, and fully connects to (i.e. *peers*) in each hour and day. In total in the *longitudinal experiment*, our machine `PING`/`PONG`s 1,301,568, starts a TCP connection with 194,608, and fully connects with 90,265 unique `ENODEID`s. We see the fluctuation of our client's peer count in Figure 4(right), where we plot the peer count over time for different runs. We can see the peer count rise steadily during the beginning of a measurement and then fluctuate, often dropping by half before picking up again. We see

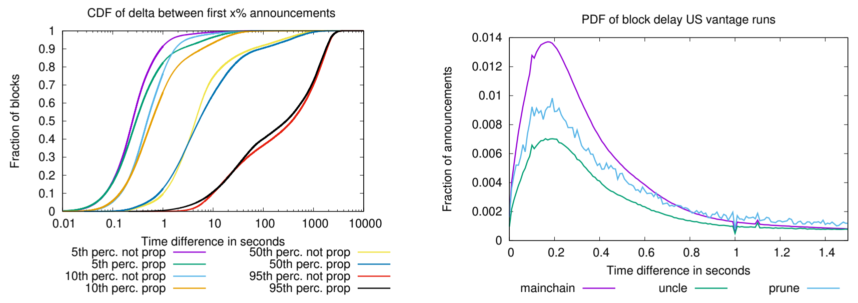


Fig. 3: *Left*: The difference in time for when our *longitudinal experiment* client hears about a block from the first announcement to all subsequent announcements by percentile. *Right*: Probability distribution of the time of subsequent announcements since our client first heard about that block from any peer.

this significant churn in our full connections in Figure 5,(left): we compare the average number of peers we have to the number of new connections and connections that end each hour, and see that within an hour we might make up to 3x the number of connections as our average connections. This means each hour, our client makes/ends around 1K-1.5K connections to 300-400 unique `ENODEIDs`, i.e. reconnecting to the same nodes.

We further explore the trend of re-connecting to the same nodes in Figure 5,(right), where we plot a breakdown of the length of connections for connected peers based on how long we are connected (*short* is 0–10 seconds, *medium* is 10–1,000 seconds, *long* is 1,000 seconds or more). We can observe that while over 80% of unique peers we connect with in an hour are long connections ($>$ 1000s), we do see around 10% of connections to both short and medium peers.

Peers who participate in block propagation Given this high level of churn, looking at *all* connections would be significantly biased by the many short connections. Thus, we focus more narrowly on the peers who actually affect information propagation in the Ethereum network: those peers who propagate blocks to our client, called `useful` peers.

We plot the count of `useful` peers in Figure 6 , plotting the number of peers who inform our client of different types of blocks each hour over the course of the run. We can observe around 400 unique peers per hour (and around 1,000–2,000 unique peers per day, not shown) who announce relevant blocks.

We dig deeper into the behavior of different peers by comparing the connection lengths of three groups of peers: all peers, `useful` peers, and first announcers (those peers who are the first to announce a block to us mined by the top 15 miners). In Figure 6 we compare connection lengths and number of days we observe these peers. We see a clear distinction between all peers and those announcing blocks to us, where the latter tends to have longer connections and show up more days. There is an increase around 1,000 seconds for average peer online times which correspond to many `ENODEIDs` coming from a few IPs (making up about 30% of the announcing `ENODEIDs`) who are online only once for about 1,000 sec-

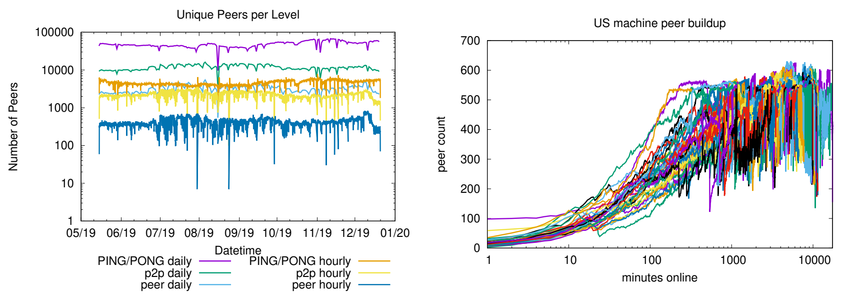


Fig. 4: *Left*: the number of unique peers we PING/PONG, start a TCP connection with, and establish a full connection with per hour and day. *Right*: Peer count per minutes online, a line for each run of the *longitudinal experiment* client, showing significant churn even after we fully join the network.

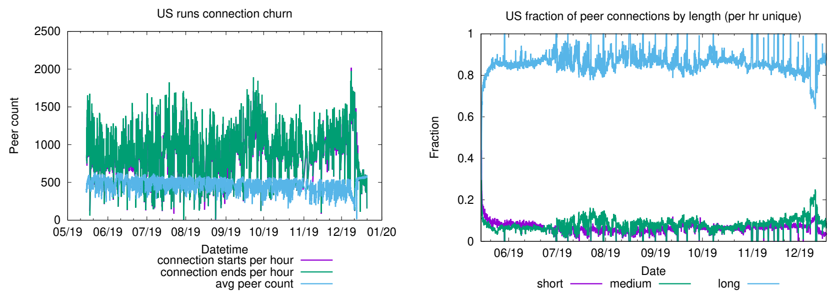


Fig. 5: *Left*: the hourly number of peer connections that we start/end (similar counts), and our average number of peers. *Right*: unique connections per hour by length: short (<10s), medium (10-1000s) and long (>1000s). Both figures together show how the majority of the churn (short connections) are due to a minority of our peers.

onds[10]. Finally, the "spike" at 15 seconds in all connection lengths comes from over 5,000 `ENODEIDs` mostly from a single IP in China which reconnects many times; it disappears when we normalize by IP in the middle graph.

Peer location Next we examine the physical location of peers by using the `geolite2` and `ip2geotools` tools to map IP addresses to continents. In Table 2, we take a closer look at where the peers our client connects to are located. We list all P2P nodes we connect to, all Ethereum peers and all `useful` peers across the entire run. To observe a snapshot as well, we also include the `useful` peers from a single 24-hour snapshot in 2019 and another in 2020. Generally the majority of our peers are in Asia, Europe and North America, with useful peers skewing more towards North America and P2P peers coming primarily from Asia.

	Fraction of peers in locations							
Data set	**Africa**	**Asia**	**Europe**	**N.Amer**	**Oceania**	**S.Amer**	**Unkn**	**Count**
P2P	0.0060	0.472	0.232	0.255	0.0148	0.0144	0.004	138,107
Ethereum	0.0057	0.348	0.283	0.329	0.0173	0.0108	0.006	55,091
`useful`	0.0014	0.315	0.262	0.398	0.0139	0.0049	0.004	22,982
2019-`useful`-24 hr	0.0010	0.289	0.286	0.409	0.0093	0.0046	0.0009	1,079
2020-`useful`-24 hr	0.0037	0.273	0.249	0.459	0.0129	0.0025	0.0006	1,632

Table 2: Fraction of peers (by unique IP address) across continents for all P2P connections, all Ethereum peer connections, and all useful peers across the entire *longitudinal experiment*. Also included are the `useful` peers from two 24 hour periods 1 year apart (05/20/2019 and 05/20/2020).

4.4 Internet location

Finally, we explore the effect of the network geographical position on how peer connections are made and how blocks are propagated. We discuss the highlights of our findings in Table 3. We first look at the location of the peer who told

[10] Mostly Coinbase nodes who appear to be routinely generating a fresh `ENODEID`.

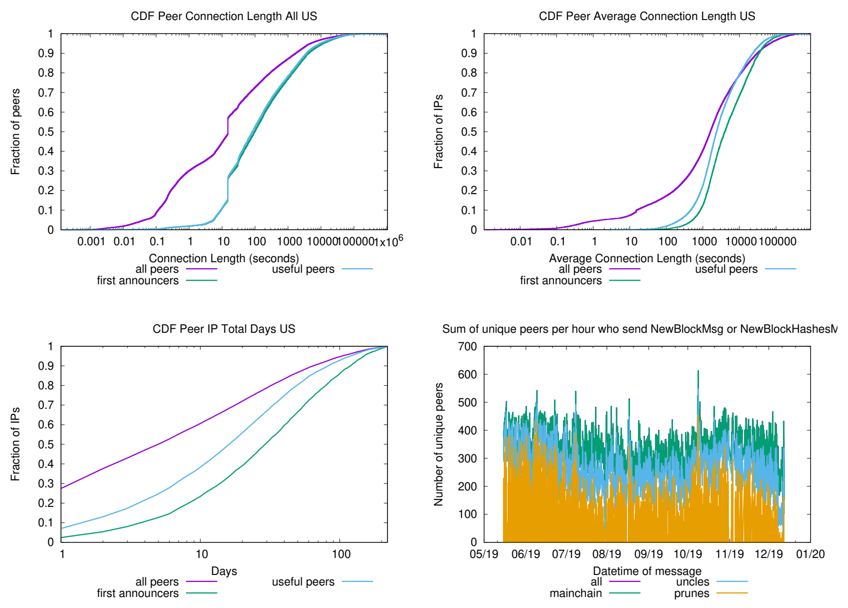


Fig. 6: *Top left*: Cumulative distribution of all connection lengths for peers. *Top right*: Average connection length when grouping peers by IP. *Bottom left*: Total number of days our client observes peers active. *Bottom right*: Number of unique peers who announce blocks to us per hour for each kind of block and total unique per hour.

our *longitudinal experiment* client about each block first. We see that our client primarily first heard blocks from peers in North America, with Europe and Asia closely following. However, it is unclear the extent to which this is due to the fact that our client in the *longitudinal experiment* was located in North America.

To explore this, we turn to examine the *multiple vantage point experiment* run, where we also run clients in parallel in Seoul and Frankfurt. We look at the fraction of where the first announcement for *all* blocks are received from by continent, and also filter for just the blocks that machine was the *first* to hear about (i.e. before the other two locations)

We can immediately observe that all three of our clients primarily hear about new blocks from peers on the same continent where they are located, and this effect is particularly strong when they are the first to hear about a block: our U.S. client hears about new blocks for the first time from a North America peer 81% of the time; our Seoul client hears about blocks first from an Asia peer 90% of the time; and our Frankfurt nodes hears about blocks first from a Europe peer 90% of the time. Additionally in 10 in Appendix D we see some evidence that the location of our machine effects the duration of it's connection to peers.

To determine which of our three nodes heard about blocks first *overall* we ensure all of our clients are synced to local timeservers using NTP. All three clients "win" roughly one-third of the time.

The results thus far suggest that the majority of peers are located in North America, Europe, and Asia (this observation is consistent with the peer information posted on `ethernodes.org` as well) and that may also be where mining

			Peer Location							
Runs		**Node**	**AF**	**AS**	**EU**	**NA**	**OC**	**SA**	**total**	**fraction**
longitudinal		Virginia	2e-5	0.099	0.158	0.742	9e-4	5e-5	393,276	
All Blocks	***multiple vantage***	Virginia	0	0.0679	0.205	0.727	0.000432	5e-5	39,345	
		Seoul	0	0.632	0.101	0.264	4e-3	3e-5	39,345	
		Frankfurt	0	0.0523	0.849	0.0989	3e-5	0	39,345	
	info prop	Virginia	4e-5	0.0469	0.118	0.834	2e-4	0	69,199	
		Seoul	3e-4	0.766	0.048	0.182	0.0038	4e-5	69,199	
		Frankfurt	1e-4	0.103	0.788	0.109	6e-5	3e-5	69,199	
		São Paulo	2e-4	0.153	0.364	0.481	0.001	4e-4	69,199	
		Sydney	9e-5	0.550	0.1006	0.346	0.0045	7e-5	69,199	
First to Hear	***multiple vantage***	Virginia	0	0.0485	0.143	0.808	2e-4	2e-4	12,244	0.311
		Seoul	0	0.897	0.00569	0.0972	2e-4	0	12,474	0.317
		Frankfurt	0	0.0144	0.896	0.0891	0	0	14,627	0.372
	info prop	Virginia	0	0.011	0.0548	0.935	0	0	5,926	0.086
		Seoul	2e-4	0.797	5e-4	0.199	0.0028	0	33,180	0.479
		Frankfurt	4e-5	0.0418	0.905	0.0532	0	0	27,888	0.403
		São Paulo	0	0.465	0.222	0.313	0	0	243	0.003
		Sydney	5e-4	0.808	0	0.188	0.0041	0	1,962	0.028

Table 3: For each experiment and machine, we look at when the node first heard about a block and the location of the peer who told us about the block. For the *multiple vantage point experiment* and *information propagation experiment*, we distinguish when each location was the first to hear about a block.

is centered(i.e. where blocks are originating from). Thus, we wanted to run an additional run with clients located *far away* from the majority of the network to see how they perceive the network. To do so, we use the *information propagation experiment* runs (which occurred about 1 year after the *multiple vantage point experiment* runs), where we now include additional nodes in Sydney and São Paulo. As before, we observe that the North America, Asia, and European clients all typically hear about blocks first from peers on their continent. However, when we examine the new vantage points, we observe that the São Paulo client receives most of its blocks first from Asia, followed by North America then Europe; the Sydney client receives the vast majority of its blocks first from Asia.

We also examine whether any of our clients are the first to tell one of the other clients about a block. We observe that though it does happen (especially from Frankfurt/US/Seoul to São Paulo and Sydney), it is a small fraction of the 69K blocks mined during these runs. For example, in the *multiple vantage point experiment*, our Seoul client is the first to inform the Virginia client, and Frankfurt is the first to inform the Seoul client about a block first *just once*, while the Virginia client tells the Seoul client about 6 blocks first.

As a final analysis, we examine how "quickly" information in the Ethereum network propagates to our different clients. See Appendix D for the plots of propagation time for `mainchain` blocks in both sets of runs. Looking at the *multiple vantage point experiment*, the difference from when the first and second machines hear about a block is less than 100 milliseconds for 85% of the blocks, and about 50 milliseconds for 50% of the blocks. However, looking at the *information propagation experiment*, we can see that the Sydney and São Paulo clients are at a clear disadvantage, with a much longer and fatter tail of incoming messages. We can see this even further in the final column of Table 3, where we see that the São Paulo client and the Sydney client are the first *overall* to hear about blocks only 0.3% and 2.8% of the time, respectively.

5 Discussion

We set out to better understand the structure of the network that powers Ethereum. How this network operates has implications both on the security of the underlying blockchain (e.g. the immutability of the blockchain) and on the experience of users who need access to the blockchain in order to interact with it (e.g. send/hear about transactions). Prior measurement work on Ethereum has focused primarily on information stored on the blockchain or on the peer discovery protocol of Ethereum. The main novelty of this work is on bridging both observations of peer connectivity behavior with the block information the peer provides.

In our *longitudinal experiment* spanning 7 months, we observe a small fraction of the nodes we connect with actually passing all the handshake checks and becoming full peers. We were able to start a TCP connection with 194,608 nodes but only ended up successfully peering with 46% of them (Table 1). Moreover, we found significant churn in the network, with more than 45% of those peers only staying connected for up to 10 seconds per connection (Figure 6). Additionally we found that not all of those peers actually tell us about blocks, only about 27% of our peers are `useful`, but they tend to stay connected for much longer time than the non-`useful` ones. Maintaining longer connections with nodes than previous work allowed us to capture propagation behavior of our peers which revealed how few of the nodes we connected to participate in block propagation. Furthermore, while examining the unexpected behavior of peers, we were unable to discover the motivation for the common practice of connecting sporadically for very short periods of time.

In the *longitudinal experiment* we were also able to examine miner behavior. Unsurprisingly we find a small subset of mining ids(primarily big known mining pools) mine the majority of blocks, but at varying efficiencies(i.e. differing ratios of `mainchain`, `uncle` and `prune` blocks). Though only about 14% of our full peers were ever first to announce a block from the top 15 miners to our client, we were unable to find any correlation between which peers are the first announcers for which pools. This is likely by design (of the miners) as tracing blocks back to miners would put them at risk for targeted attacks. Though we do find that first announcers do maintain longer connections than even the `useful` peers.

In order to connect the behavior of peers to the speed with which information propagates through Ethereum's peer-to-peer network, we looked at how long it takes for our peers to tell us about a block after it is mined (Figure 3).

Though the propagation delay distribution peaks at around 20ms, `prunes` and `uncles` have a longer and heavier tail. This corresponds to `mainchain` blocks winning block races. Looking deeper into the tail end of the propagation distribution we do find a variety of odd behavior for all blocks. It is often the case that blocks continue to be announced to us by new peers for hours after the first announcement, and even after we have announced the same block to those peers. It is difficult to speculate whether this behavior is malicious or from slow machines/connections. Protocol changes to filter this behavior could thus inadvertently penalize clients with slow connections or hardware who may be trying to honestly participate in the network. We find additional odd behavior with the majority of `prune` blocks being advertised as having block numbers vastly deviating from the `mainchain` height, and being advertised by just a handful of peers. As these are sent mostly as new hash announcements (and not the full block), they are likely cheap spamming behavior and should potentially be filtered by the protocol.

Lastly, by running nodes in several parts of the world, we found that the location of the node has an effect on when it hears about blocks first and where it hears them from. Moreover, miners do appear to stand to gain an advantage by operating out of specific locations that hear about blocks sooner. Our work suggests that there may be significant locations at a disadvantage, so the extent of this should be further studied and its implications on the decentralization of the network. A finer appraisal of delay can also be done by observing transaction traffic, as there are significantly more transactions flowing through the network than blocks. Though it is known that transactions can impact consensus [10], actually linking them to miner behavior is significantly more challenging than blocks as blocks must originate from the miner but transactions can generally come from any user in the network. Additionally we believe that understanding the sporadic behavior of peers and the behavior of those peers not involved in block propagation is key to understanding the health of Ethereum's P2P network and should be a focus for future work. Previous work on Eclipsing attacks on both Ethereum and Bitcoin networks have shown how high churn in the network aids an adversary in their attack [21, 22, 29]. Extending connection timeouts to avoid early disconnects is a counter measure they prose, and based on our observations could aid in preventing a portion of our disconnects. Though the experiments we run are resource intensive (e.g. memory and bandwidth to maintain many connections), they can be extended to any P2P network of other cryptocurrencies, changing only the peer cap limit to be sufficient to hit a representative portion of the network. As such, comparing our results to the behavior of other networks would be illuminating.

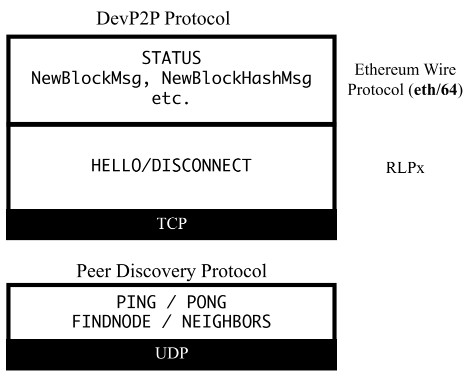
References

1. Ethash. https://github.com/ethereum/wiki/wiki/Mining#ethash-dag.
2. Ethereum wire protocol (eth). https://github.com/ethereum/devp2p/blob/master/caps/eth.md.
3. Measuring ethereum nodes. https://medium.com/coinmonks/measuring-ethereum-nodes-530bfff08e9c.
4. Node discovery protocol. https://github.com/ethereum/devp2p/blob/master/discv4.md.
5. The rlpx transport protocol. https://github.com/ethereum/devp2p/blob/master/rlpx.md.
6. Ethereum market capitalization. https://coinmarketcap.com/currencies/ethereum/, 2020.
7. Ethereum node explorer. ethernodes.org, 2020.
8. L. Anderson, R. Holz, A. Ponomarev, P. Rimba, and I. Weber. New kids on the block: an analysis of modern blockchains. *arXiv preprint arXiv:1606.06530*, 2016.
9. M. Bartoletti, S. Carta, T. Cimoli, and R. Saia. Dissecting ponzi schemes on ethereum: identification, analysis, and impact. *Future Generation Computer Systems*, 102:259–277, 2020.
10. P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*, 2019.
11. C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.
12. J. A. D. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí. The bitcoin p2p network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.
13. N. El Ioini, C. Pahl, and S. Helmer. A decision framework for blockchain platforms for iot and edge computing. SCITEPRESS, 2018.
14. Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao, and Z. Yin. Topology measurement and analysis on ethereum p2p network. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE, 2019.
15. A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer. Decentralization in bitcoin and ethereum networks. In *International Conference on Financial Cryptography and Data Security*, pages 439–457. Springer, 2018.
16. A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16. ACM, 2016.
17. A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705, 2015.
18. go-ethereum client. `https://github.com/ethereum/go-ethereum`.
19. R. Greene and M. N. Johnstone. An investigation into a denial of service attack on an ethereum network. In *Proceedings of the 16th Australian Information Security Management Conference*, page 90, 2018.
20. E. E. Group. Networking: Dev p2p, rplx, discovery, and eth wire protocol: via zoom: https://www.youtube.com/watch?v=hnw59hmk6rk.

21. E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 129–144, 2015.
22. S. Henningsen, D. Teunis, M. Florian, and B. Scheuermann. Eclipsing ethereum peers with false friends. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 300–309. IEEE, 2019.
23. C. Holotescu et al. Understanding blockchain opportunities and challenges. In *Conference proceedings of» eLearning and Software for Education «(eLSE)*, volume 4, pages 275–283. " Carol I" National Defence University Publishing House, 2018.
24. M. Imamura and K. Omote. Difficulty of decentralized structure due to rational user behavior on blockchain. In *International Conference on Network and System Security*, pages 504–519. Springer, 2019.
25. M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis. Churn in the bitcoin network: Characterization and impact. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 431–439. IEEE, 2019.
26. L. Kiffer, D. Levin, and A. Mislove. Stick a fork in it: Analyzing the ethereum network partition. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 94–100, 2017.
27. L. Kiffer, D. Levin, and A. Mislove. Analyzing ethereum's contract topology. In *Proceedings of the Internet Measurement Conference 2018*, pages 494–499, 2018.
28. S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, and M. Bailey. Measuring ethereum network peers. In *Proceedings of the Internet Measurement Conference 2018*, pages 91–104, 2018.
29. Y. Marcus, E. Heilman, and S. Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. *IACR Cryptol. ePrint Arch.*, 2018:236, 2018.
30. S. B. Mariem, P. Casas, M. Romiti, B. Donnet, R. Stütz, and B. Haslhofer. All that glitters is not bitcoin–unveiling the centralized nature of the btc (ip) network. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.
31. P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.
32. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008.
33. F. Victor and B. K. Lüders. Measuring ethereum-based erc20 token networks. In *International Conference on Financial Cryptography and Data Security*, pages 113–129. Springer, 2019.
34. G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

Appendix

A Ethereum's Networking Stack



Here we show a breakdown of Ethereum's networking stack: The two protocols, Peer Discovery and DevP2P, run in parallel, with the former feeding new peers to the latter to form connections with.

The Discovery protocol `discv4` [4] is a UDP protocol based on the Kademlia [31] distributed hash table (DHT) that allows nodes to bootstrap and find other nodes. Nodes generate an ECDSA key-pair, with the 512-bit public key acting as their unique identifier (`ENODEID`).

To bootstrap, a node starts querying peers it already knows about from previous runs—or the bootstrap nodes, if it knows of no such nodes—for the other nodes. The client sends a FINDNODE query to learn about other nodes, who respond with a set of peers whose `ENODEID` is closest to the requestor's `ENODEID`. In lieu of Kademlia's conventional XOR distance metric, Ethereum uses its `logdist`[29] distance metric, on the hash of the `ENODEID`.

The Discovery layer is meant to act as a general-purpose layer for multiple different networks to discover each other.[11] Gao et al. [14] measured an average of over 70,000 active nodes per day in this layer; of those only roughly 10,000 were running the Ethereum-Application layer protocol mentioned below. The upshot is that just because a node exists in the Discovery layer does not mean it is running Ethereum.

B Block Propagation Algorithm

A simplified reproduction of how a block is handled and propagated when `geth` receives it.

[11] other protocols include SWARM, LEA, Whisper or even the Ethereum protocol with different chain IDs or genesis hash.

Algorithm 1: Block handling and Propagation Algorithm

```
Input: $block$ from network, $blockchain$ from storage
$peer \leftarrow block$.sender $peer$.has[$block$.hash] = TRUE if $peer$.pendingBlocksCount $> 64$ then
    return
$heightDiff \leftarrow block$.number $-$ $blockchain$.chainHeight if $32 < heightDiff <$ -7 then
    return
if $block$.parent.hash not in $blockchain$.blocks then
    return
if $block$.verifyHeader() == $False$ then
    return
propagate($block$) /* send entire block */
$blockchain$.insert($block$) announce($block$) /* send just block hash */
```

Algorithm 2: block.VerifyHeader()

```
Input: $block$, $blockchain$
$header \leftarrow block$.header $parent \leftarrow blockchain$.getParent($block$) if $header$.Time $>$ (Now() + 15 seconds) then
    return False
if $header$.Time $\leq$ $parent$.Time then
    return False
$expectedDifficulty \leftarrow blockchain$.CalcDifficulty($header$.Time, $parent$) if $expectedDifficulty \neq header$.difficulty then
    return False
if $header$.gasLimit $> 2^{63} - 1$ then
    return False
if $header$.gasLimit $< 5000$ then
    return False
if $header$.gasUsed $>$ $header$.gasLimit then
    return False
if $block$.number $\neq$ $parent$.number $+1$ then
    return False
return True
```

Algorithm 3: propagate()

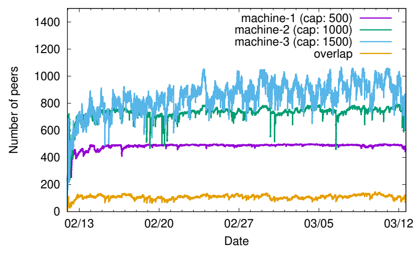
```
Input: $block$, $allPeers$
$peers \leftarrow allPeers$.withoutBlock($block$.hash) $transferLen \leftarrow max(\lfloor\sqrt{peers.length}\rfloor, 4)$ $transferLen \leftarrow min(transferLen, peers.length)$ $propagatePeers \leftarrow peers[0{:}transferLen]$ for $peer$ in $propagatePeers$ do
    $asyncSend(block, peer)$ $peer$.has[$block$.hash] = TRUE
```

Algorithm 4: announce()

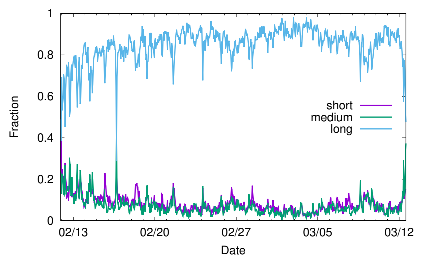
```
Input: $block$, $blockchain$, $allPeers$
1 $peers \leftarrow allPeers$.withoutBlock($block$.hash) if $block$.hash not in
    $blockchain$.blocks then
2     return
3 for $peer$ in $peers$ do
4     $asyncSend$($block$.hash, $peer$) $peer$.has[$block$.hash] = TRUE
```

C Peer Cap Experiment

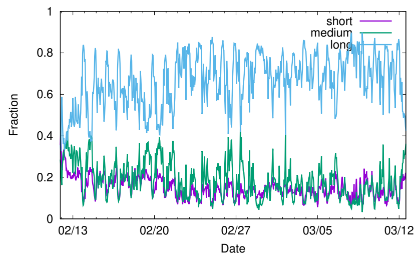
To establish if our 750 peer cap is representative of the network, we ran an experiment with parallel clients of peer caps of 500, 1000, and 1500 peers in the `us-east-1` Amazon data center (in Virginia, U.S.). We ran this experiment from February 11, 2020 to March 13, 2020, and we refer to it as the *peer cap experiment*. In Figure 7 we see that the 500 peer cap machine is able to reach



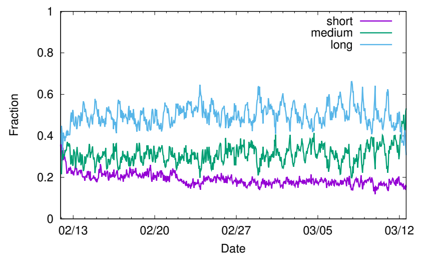
(a) Instantaneous peer count



(b) 500 peer cap



(c) 1000 peer cap



(d) 1500 peer cap

Fig. 7: We look at the instantaneous peer count for each of the peer cap runs, and a breakdown of the peers by short (<10s), medium (10-1000s) and long (>1000s) connection lengths.

the cap while the machines with higher caps are not. Likely for the 1000 peer cap, this is because it is being capped by their inbound connection cap. For the 1500 peer cap, its likely that the machine has maxed the number of peers its able to connect to given available peers.

To test this, we ran another client with a higher peer cap (3500). In Figure 8 we show the number of instantaneous peers we had over time, split by whether we initiated the connection(outbound) or the peer did(inbound). We note our peers are primarily inbound and that even with a 3500 (2.3K inbound) peer cap, our client maxed out at 1K peers.

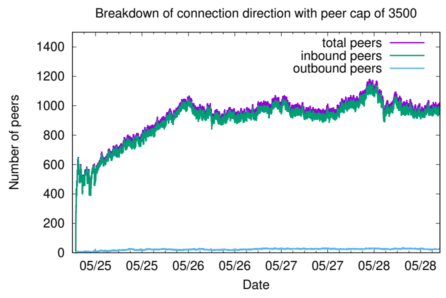
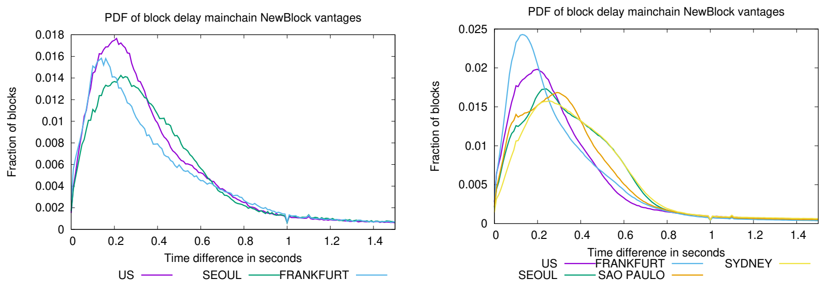


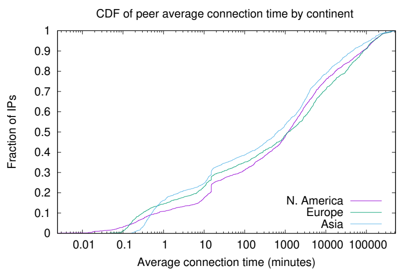
Fig. 8: We started a node with a cap of 3500 peers. The plot shows the total peers we were connected to at any time, along with the instigator of the connections. We made outbound connections to our peers, while inbound connections are when our peers connected to us.

D Effects of Location on Peers and Propagation

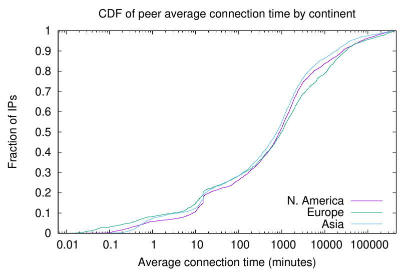


(a) *multiple vantage point* **(b)** *information propagation*

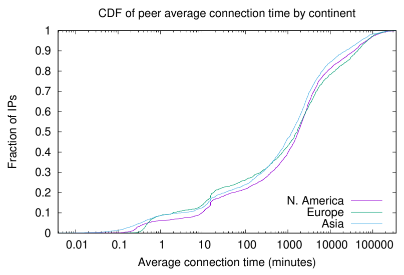
Fig. 9: Probability distribution for the time between first hearing about a block and all subsequent `NewBlockMsg` messages for *multiple vantage point experiment*and *information propagation experiment*.



(a) Virginia



(b) Frankfurt



(c) Seoul

Fig. 10: Average connection length per peer IP for each machine of the *multiple vantage point experiment* broken down by peer continent (for N.America,Europe and Asia).

E Ethernodes

Ethernodes [7] is a public web site that reports on aggregate statistics for the Ethereum network. Between March 30 and October 15th, 2019, we scraped Ethernodes twice a minute for the Ethereum node information they report from their crawler. Specifically, Ethernodes publishes a list of all nodes who they have reached in the last 24 hours (including enode id, IP address, time of last contact, and other information); this list has often been cited as an estimate of the size of the Ethereum network at any given time. We note that Ethernodes does not provide details for their data collection methodology, and they did not respond to our inquiries. We do note that during our experiments, we observed that some nodes in Ethernodes' list respond multiple times per minute while our clients show up in their data as responding around every 10 minutes. Regardless, we will use the Ethernodes data as a point of comparison for our data analysis.

Ethernodes is widely cited when reporting statistics about the Ethereum network, including in academic work [16, 19, 13, 24, 23]. Because it is widely cited when trying to understand Ethereum's network, we felt it would be useful to compare our data to theirs. Previous work [28, 3] has claimed that Etherenodes incorrectly counts Ethereum peers by publishing many nodes who fail some part of the DevP2P handshake often because they are running the Ethereum Classic

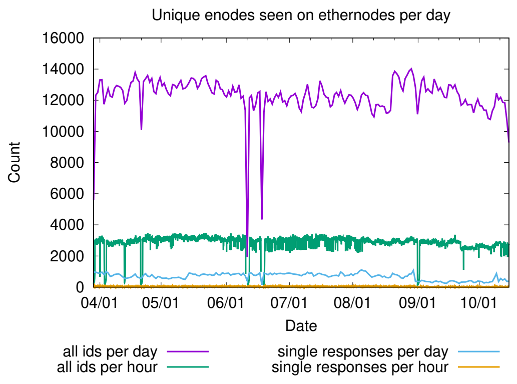


Fig. 11: Total unique `ethernodes.org` nodes seen in a day and in an hour. We observe many nodes that show up only one day and never again in the span of the measurement study, we plot those as "single responses" by the total that appear in a day and those broken down by hour. There is an average of about 780 of these per day and 40 per hour.

or other fork protocol. To dig deeper, Figure 11 shows the number of unique `ENODEIDs` that show up in the logs per day and per hour as well as the number of ids that we see show up only one day and that number broken down by hour. Overall, `ethernodes` reports around 12,000 unique `ENODEIDs` per day, while only about 3,000 per hour. Almost 800 per day of these are ids that only show up that day and never again, which adds up to about 68% of all `ENODEIDs` we see in our 200-day scraping period (a third of which came from a single IP address). Similarly, we see a total of 171,241 IP addresses, 30,376 of which correspond to `ENODEIDs` which show up only once. Overall, this data is consistent with a large amount of churn, and that even if we discount some of the `ethernodes` data as not being part of the main network, the counts are still a misrepresentation of the size of the *actual* or *effective* Ethereum network at any given moment.