



HAL
open science

Ethereum's Peer-to-Peer Network Monitoring and Sybil Attack Prevention

Jean-Philippe Eisenbarth, Thibault Cholez, Olivier Perrin

► **To cite this version:**

Jean-Philippe Eisenbarth, Thibault Cholez, Olivier Perrin. Ethereum's Peer-to-Peer Network Monitoring and Sybil Attack Prevention. *Journal of Network and Systems Management*, 2022, Special Issue on Blockchains and Distributed Ledgers in Network and Service Management, 30 (4), pp.65. 10.1007/s10922-022-09676-2 . hal-03777454

HAL Id: hal-03777454

<https://inria.hal.science/hal-03777454>

Submitted on 14 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ethereum’s Peer-to-Peer Network Monitoring and Sybil Attack Prevention

Jean-Philippe Eisenbarth¹, Thibault Cholez¹, and Olivier Perrin¹

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
`firstname.lastname@loria.fr`

Abstract. Public blockchains, like Ethereum, rely on an underlying peer-to-peer (P2P) network to disseminate transactions and blocks between nodes. With the rise of blockchain applications and cryptocurrencies values, they have become critical infrastructures which still lack comprehensive studies. In this paper, we propose to investigate the reliability of the Ethereum P2P network. We developed our own dependable crawler to collect information about the peers composing the network. Our data analysis regarding the geographical distribution of peers and the churn rate shows good network properties while the network can exhibit a sudden and major increase in size and peers are highly concentrated on a few ASes. In a second time, we investigate suspicious patterns that can denote a Sybil attack. We find that many nodes hold numerous identities in the network and could become a threat. To mitigate future Sybil attacks, we propose an architecture to detect suspicious nodes and revoke them. It is based on a monitoring system, a smart contract to propagate the information and an external revocation tool to help clients remove their connections to suspicious peers. Our experiment on Ethereum’s Test network proved that our solution is effective.

Keywords: Blockchain · Security · Network Measurement · Distributed Hash Table

1 Introduction

With the rise of cryptocurrencies and the development of new services, blockchains are becoming essential systems. Without any central authority, permissionless blockchains like Ethereum solely rely on their distributed algorithms to prevent attackers to disturb the system. While proof-of-work and proof-of-stake mechanisms (as well as the CASPER protocol [1] for Ethereum) have been proven robust to prevent an attacker to forge arbitrary blocks [2,3,4,5], the underlying peer-to-peer (P2P) network got far less attention so far and creating Sybils at the Distributed Hash Table (DHT) level is still extremely easy since generating an ID is costless. We will show in this paper that some peers carry thousands of ID in the DHT address space. This may be a less critical part of the whole Ethereum architecture, but it is nonetheless important to ensure the good connectivity of the network, more precisely transmission of blocks and transactions.

Beyond security considerations, previous studies on the Ethereum P2P network lack a well described and reproducible methodology to produce their datasets. Moreover, no study so far analyzed the network with a focus on the suspicious behaviors that can threaten the proper functioning of structured P2P networks. Producing new knowledge about the Ethereum P2P network is important as it is but not sufficient as potential weaknesses should also be mitigated. That is why we propose, in a second time, a novel architecture to prevent possible attacks caused by suspicious nodes.

Our contributions are thus manifold. To begin with, we designed and implemented the first crawler for the Ethereum P2P network which code is open-source and methodology assessed. Secondly, we created two datasets containing one month of measurements conducted in September 2021 when the network was stable for several months, and between February 15th–March 7th 2022 when we observed a sudden rise in the number of nodes. We made them publicly available, and we performed their analysis by considering a few metrics that can impact the network reliability. In particular, we are the first to consider and enumerate suspicious nodes. Our third contribution is an architecture we designed and implemented to prevent Sybil attacks by revoking suspicious nodes in a safe and distributed manner.

The rest of the paper is organized as follows. Section 2 presents the background on the Ethereum P2P network and its protocols. Then, Section 3 surveys the previous work conducted on this network as well as more general questions concerning the security of similar P2P networks. In Section 4, we present and validate our crawling strategy to constitute a dataset. Its analysis is conducted in Section 5 with a special focus on high-level network properties and the detection of suspicious nodes. We describe in detail in Section 6 our architecture to revoke suspicious nodes and prevent attacks to disturb the network. Finally, Section 7 discusses another possible architectural choice and Section 8 concludes the paper.

2 Background on Ethereum’s P2P network

The Ethereum developers maintain a project named DEVp2p [6], which is the specification of the different protocols used in the Ethereum P2P network. The official client Go-Ethereum (also called Geth) along with other clients, notably OpenEthereum (formerly Parity-Ethereum), implement this specification [7,8]. In Figure 1, we give an overview of a classical interaction between two peers that rely on the three following protocols : Node Discovery, RLPx and Ethereum Wire protocol. In this study, we will focus on the Node Discovery Protocol which is used by our crawler to contact the peers and make their inventory.

The Node Discovery Protocol is based on a modified implementation of the Kademlia DHT [9]. The main advantages of Kademlia are its simplicity of implementation and its very efficient and scalable routing algorithm which complexity is in $O(\log N)$, N being the number of nodes in the network. We will

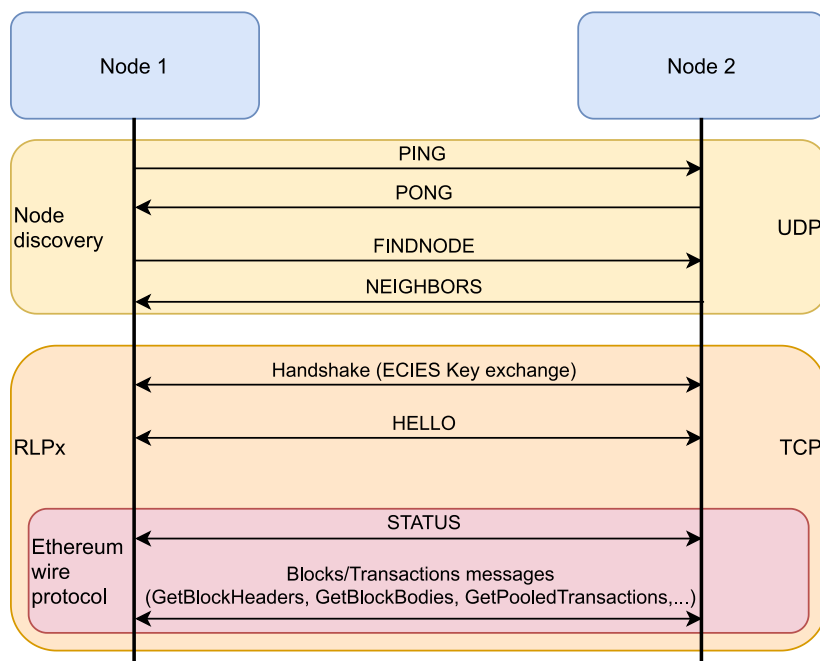


Fig. 1. Overview of the Ethereum protocols interaction between two peers

first present the main principles of Kademlia and then highlight the differences in the Ethereum implementation.

The purpose of Kademlia is to efficiently retrieve data (values) that are distributed among the peers thanks to their keys (hashes). A peer is represented by a randomly generated Node ID, and stored data is represented by its hash (for instance using the SHA-1 algorithm), both are 160-bit values. Kademlia uses a XOR-metric to compute a distance between two nodes or between a node and a data within the DHT address space. To be scalable, a peer's routing table only stores a limited number of nodes (contacts) organized in K-buckets (group of K contacts), each bucket covering a part of the ID space following a logarithmic distribution (the next K contacts cover half of the space compared to the previous bucket). This structure ensures that any ID (peer or data) can be found in a limited number of hops. Data can be stored on nodes which distance is inferior to a threshold (i.e. within a tolerance zone). To cope with churn (i.e. peers' continuous arrival and departure), data should also be replicated on several nodes. Finally, a node can request data to the nodes responsible for its storage.

Like Kademlia, Ethereum's Node Discovery Protocol is based on UDP and uses the XOR metric to compute the distance between peers. But there are significant differences between the original Kademlia design and the Ethereum implementation of the DHT. First, and most importantly, Ethereum does not use the DHT for any data storage/retrieval purposes, but instead, all the peers

store the entire blockchain. Ethereum’s DHT is only used for node discovery. Also, a peer is identified by a 512-bit ECDSA (Elliptic Curve Digital Signature Algorithm) public key (compared to a 160-bit ID in Kademlia) and the XOR-metric is applied on the hash of this key. The hash function that is used is Keccak256 and not the SHA3 FIPS 202 standard from the National Institute of Standards and Technology (NIST) which is based on the Keccak algorithm. Essentially the standard differs on the padding parameters, hence giving two different hashes.

Concerning the network primitives, there are 4 types of packets used to implement the Node Discovery Protocol: PING, PONG, FINDNODE and NEIGHBORS. When a PING message is received, the recipient should reply with a PONG message. The PING-PONG exchange is also used to obtain an “endpoint proof”: to prevent spoofing and traffic amplification attacks, a node should only reply to a sender that has been verified responsive. The node discovery mechanism is built around the FINDNODE packet: it requests information about nodes close to a given parameter `target` ID. The recipient of such message should reply with a NEIGHBORS packet containing the sixteen closest nodes to the target known from its own buckets. It is worth to be noted that the ID used in packets of the Node Discovery Protocol is the 512-bit ECDSA public key instead of its 256 bits hash counterpart, although the first action of the receiver is to compute the hash. This is not optimal from the network perspective. Like Bitcoin, transactions and blocks are exchanged following a gossip protocol thanks to another level of unstructured overlay network made of direct TCP connections established between peers (see RPLx layer in Figure 1), without any consideration for their place in the DHT.

To finish, here are definitions that we will use in the rest of this paper. We will use the term `node` to refer to the association of an IP address and a port. The `NodeID` is the generated public key of a node (hexadecimal format). And an `enode` is the association of a `NodeID`, an IP address and a port (`nodeid@ip:port`). We will also use a few definitions to describe the state of an Ethereum node. A node can be either reachable, which means public and routed to the Internet, or unreachable. An unreachable node could simply be offline or it could be online but not publicly accessible (behind a Network Address Translation (NAT) or a firewall, for example). In the latter case, it can still participate in the network via outgoing connections but it will not accept inbound connections and only the nodes to which it has initiated a connection know that it is online.

3 Related Work

The originality of Ethereum, namely the Turing-complete programming language Solidity that allows programmers to write smart-contracts that are executed in a decentralized manner, has made this blockchain system very popular. There are more than one million transactions that are exchanged on a daily basis and the P2P network can be challenged, and even more when considering malicious behaviors. Thus, in addition to the research works focusing on the protocols and

the underlying P2P network used by the Ethereum’s blockchain, there are also studies of the overall security of the P2P network against specific attacks such as Sybil or eclipse attacks that could be leveraged to attack the blockchain.

3.1 Ethereum’s P2P Network Measurement

There are works that focus on the P2P network and its characteristics, as it is the case for the work of Kim et al. [10]. They developed a measurement tool called NodeFinder for characterizing the peer ecosystem. It is based on a modified version of Geth. They revealed that, in 2018, the Ethereum P2P network is noisy, i.e. there are different protocols and subprotocols that share the same discovery protocol and fewer than half of the peers contributes to the main Ethereum blockchain. Among Ethereum nodes, there are official and unofficial clients running stable and unstable versions and the size of Ethereum is significantly smaller than other public P2P networks such as Gnutella (15,425 nodes seen versus 62,586 in a 24 hour period).

Gao et al. [11] confirm the findings of the authors of NodeFinder. In 2019, they deployed several Ethereum clients (Parity light nodes) to gather information about the peers. They found that the network is cluttered by nodes that do not participate in the Ethereum wire protocol. Among the active nodes, they could find 11,000 nodes that accept ingress connections (defined as routable nodes).

In 2020, Maeng et al. [12] deployed a modified Geth node for 24 hour to collect data from the peers of the network. Their tool discovered 8,223 peers which are mainly distributed in the United States of America, China and Germany (more than 75%). Also, the peers are mainly running the Geth and Parity Ethereum clients. They also used their tool to gather the information of the neighbours of the peers and infer the topology of the network along with node degree (number of outgoing connections). To the best of our knowledge, Ethereum uses a DHT for the peer discovery process (Section 2) and this information cannot be used to infer real (TCP) connections between peers (being neighbours in the DHT does not mean that there exists an edge in the P2P network between them).

Concerning the overlay topology, Li et al. [13] gathered in 2020 a dataset based on passive monitoring of Node Discovery traffic at the border routers of an ISP (Internet Service Provider) in order to visualize the topology of the P2P network. They validate this dataset by assessing the availability of the discovered node through TCP connections. They found that the network is composed of about 10,000 nodes. They generated a topology visualization graph that shows that most of the nodes have a small degree and excellent connectivity. They also showed that the nodes gathered belong to approximately 1,400 ASes (Autonomous Systems), the top ten of them host about 60% of the network. This denotes that the Ethereum network is very concentrated and vulnerable to routing attacks, such as BGP (Border Gateway Protocol) hijack.

Also in 2020, Wang et al. [14] developed a tool called Ethna (Ethereum Network Analyzer) that is composed of two programs: NetworkObserverNode and LocalFullNode. The first one is set to only receive the blocks but not

to verify/propagate them (fast synchronization mode) and the latter is normal Ethereum node that receive, verify and propagate the blocks and transactions. They used `NetworkObserverNode` to gather measures of the transaction-propagation time and total number of transactions forwarded by nodes. From these measures, they derived features such as nodes degree and transactions broadcast latency. `LocalFullNode` is used to validate their methodology. They found that the average degree of Ethereum nodes is 47, but a few nodes have a degree greater than 1000. They also found that the messages within the network are broadcasted to most of the Ethereum nodes within 6 hops. This denotes that the Ethereum P2P network has the property of small-world: nodes are connected to a small subset of the entire nodes but most nodes can be reached in a few hops.

Unfortunately, those previous studies did not assess their crawler accuracy nor made their crawler or dataset publicly available which make it impossible to build upon them. That is something we aim to correct in this study.

3.2 P2P Network Security

P2P network security has been a vast subject a decade ago. The main well-known threat to public P2P networks is the Sybil attack [15] where a given entity creates many peers to gain the control over a P2P network. Several concrete applications can be performed once Sybil nodes are inserted in the network, like network partition or eclipsing data [16]. Such attacks were successfully launched on KAD, which is another large-scale public P2P network based on Kademlia [17,18,19,20]. Steiner et al. [17] injected 2^{16} Sybils from a single computer in a small zone of the DHT, so that they were able to catch most of lookup requests for data indexed within this zone. Wang et al. [18] managed to partition the DHT and to do massive denial of service attacks with few resources by overwriting the IP address of legitimate contacts in peers' routing table. Kohonen et al. [19] and Cholez et al. [20] managed to control the lookup process respectively by generating Sybils progressively closer to the target until the lookup process stops with a timeout, or by inserting a few distributed peers closer than any legitimate peer to the targeted DHT entry. Even though protection mechanisms have been proposed to mitigate these Sybil attacks [21], they are not all implemented in the Ethereum P2P network and the fact to derive a Sybil's place in the DHT from a ECDSA key generation followed by a hash is not costly enough to prevent large or localized attacks.

A few closer studies considered the security of the Ethereum P2P network itself. In [22], the authors made a comprehensive study of how the Ethereum network and protocols work. They also proposed two interesting techniques to carry an eclipse attack on a peer. The principle of these techniques is to monopolize the incoming and outgoing connections of the victim. They proposed seven countermeasures to prevent the eclipse attack, two of them have been implemented in Geth v1.8.0. Essentially, for a node it consists in waiting for the

seeding phase¹ to finish before accepting unsolicited PING and adding them to its buckets. Xu et al. in [23] proposed a detection model for these specific techniques. It is based on a random forest classification that is trained on the UDP packets used to carry the eclipse attack. More than 90% of the malicious packets can be correctly identified, preventing the eclipse attack. We can see that the P2P network of Ethereum is challenged by researchers but, beyond proofs of concept of attacks, we lack a comprehensive study and a global solution to prevent Sybil attacks at the network scale.

To summarize, previous measurement studies lack a well documented and reproducible methodology to produce their datasets. In particular, the datasets and tools they used are not available. Moreover, to the best of our knowledge, no study so far analyzed the P2P network with a focus on the suspicious peers that could threaten the proper functioning of structured P2P networks. On the opposite, the tools we developed in our work and the datasets we made are open-source and publicly available, making our results fully reproducible. Moreover, analyzing Ethereum’s P2P network with a focus on suspicious peers and providing a way to revoke them is totally new.

4 Measurement Strategy

In this section, we will detail how we crawl the Ethereum P2P network, which tools we used and how we validated the consistency of our results.

4.1 Crawling Methodology

We developed a crawler, named Crawleth, to crawl the Ethereum P2P network. Crawleth works at the Node Discovery protocol level. It is written in Python and partly based on the Trinity Ethereum client². Its source code is open and publicly available on the Gitlab of our research institute [24]. The goal of Crawleth is to find all the nodes following Ethereum’s Node Discovery protocol regardless of the protocol used above: it can find the nodes of the Ethereum’s mainnet and testnet, along with the nodes that participate in other blockchain systems built on top of the Ethereum’s Node Discovery layer. But previous studies [10] have shown that the majority of the peers following the Node Discovery protocol actually belongs to Ethereum’s mainnet.

It is important to note that the protocol has a countermeasure against any traffic amplification attack: it verifies that a sender of a packet participates in the discovery protocol. The sender is considered verified if it has answered to a PING within the last 12 hours. So, our crawler constantly listens to PING messages and responds with the corresponding PONG.

¹ It consists in the bootstrap of the buckets: a node contacts randomly selected peers chosen in the `db` structure, which is the database stored on disk that contains information of all the nodes ever seen, and sends `FINDNODE` packets to populate its buckets.

² <https://github.com/ethereum/trinity>

The crawling methodology of our software is depicted in Figure 2. First, Crawleth tests the connectivity of the bootstrap nodes (which IP addresses and Node IDs are retrieved from the Geth client source code where they are hard-coded) by sending a PING packet and waiting for the PONG response. Then, for each responding node, it sends a FINDNODE packet with the target parameter being the bootstrap Node ID itself. The FINDNODE requests used by our crawler are always centered on the contacted Node ID, where its routing table has the highest precision because the buckets are the deepest (more contacts are known). The recipients will answer with a NEIGHBORS packet containing sixteen close nodes to itself, in terms of Node ID XOR distance. Crawleth will carry on this strategy on the newly discovered nodes until there is no new nodes to discover, meaning it went all around the DHT. When a crawl is finished, it exports the information of the discovered up nodes (IP address, UDP port, Node ID, geolocalization) and starts a new crawl.

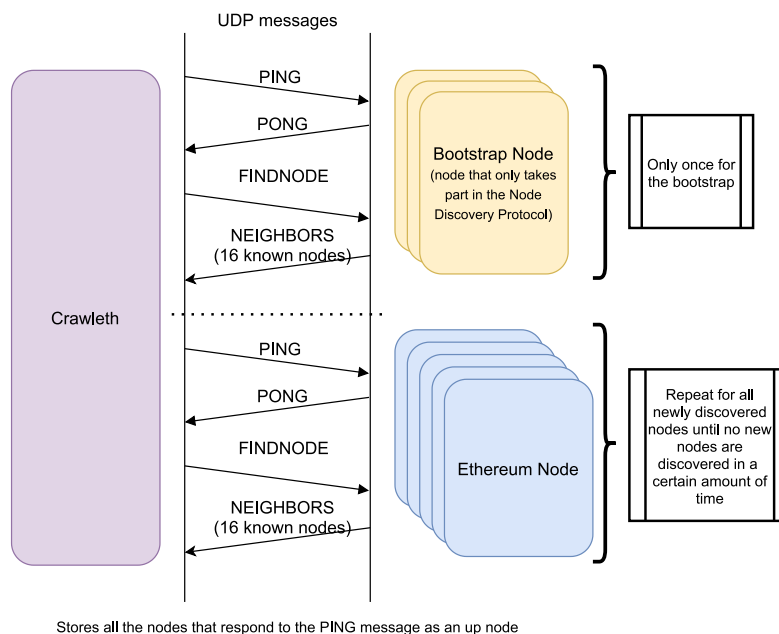


Fig. 2. Crawling strategy

4.2 Technical Setup

The machine we used is an Ubuntu 18.04 LTS computer with an Intel Xeon Processor E5-2420 v2 6 cores, 16GB RAM and a 1 Gb/s network link. To handle the large number of network connections needed, we had to increase the opened

files limit of our GNU/Linux system to 1,000,000. The machine operates the crawling node 24 hours a day, 7 days a week.

It is worth mentioning that our infrastructure lacked an IPv6 link and TOR proxy, thus all the discovered nodes are IPv4 nodes. Nevertheless, as we will see in the Section 4.3 other tools like Ethernodes or Etherscan NodeTracker do not integrate such nodes either. Concerning TOR, it seems impossible at first glance to find a node using the TOR network as TOR transports data over TCP and Ethereum needs UDP for node discovery. Although it is possible to manually bootstrap a node with known peers of the network and disable the discovery mechanism, it is very unlikely to happen since this workaround requires a tedious manual management of contacts to keep the node connected, what is normally devoted to the Node Discovery Protocol.

4.3 Validation

In order to validate the correctness and quality of our tool, we will detail in the next two subsections the convergence measurements we made and compare the coverage of Crawleth with independent external sources.

Internal Validation Crawling the DHT by requesting successively all nodes in the address space should converge in a remarkable way with a quite constant discovery rate of new nodes during a crawl while the crawler moves forward on the ID space and, at the end, a sudden drop of the discovery rate when all the ID space has been covered and there is no new nodes left to discover. As we can see in Figures 3 and 4, there are actually three phases during a crawl. The first phase is short (less than 1 minute) and is considered as a bootstrap phase where the crawler discovers a high number of new nodes per second. Then, the main phase lasts for almost all the crawl and exhibits a steady grow where it discovers a constant average number of new nodes per second (linear growth in Figure 3) while the crawler progresses through the DHT ID space. Finally, and as expected, it finishes by a last short phase where there is no new node to discover.

Furthermore, we deployed five independent ground-truth nodes running the unmodified official Go Ethereum client [7] (Geth v1.10.3, released in May 2021). One of them was configured to join the Ethereum Testnet Ropsten (networkid 3) and the four others were configured to join the Ethereum Mainnet (networkid 1). All the other networking settings have been left by default, in particular the maximum number connections to other peers which is 50. All the nodes were deployed on the same server but this does not affect their connections since the node discovery uses an overlay network (DHT) which is totally agnostic to geographic or IP level considerations. They all have different NodeIDs, thus different place in the DHT address space. When they are connecting to the P2P network, they all received different responses from the bootstrap nodes. We also verified that they have different active connections. The crawler was able to find all of them.

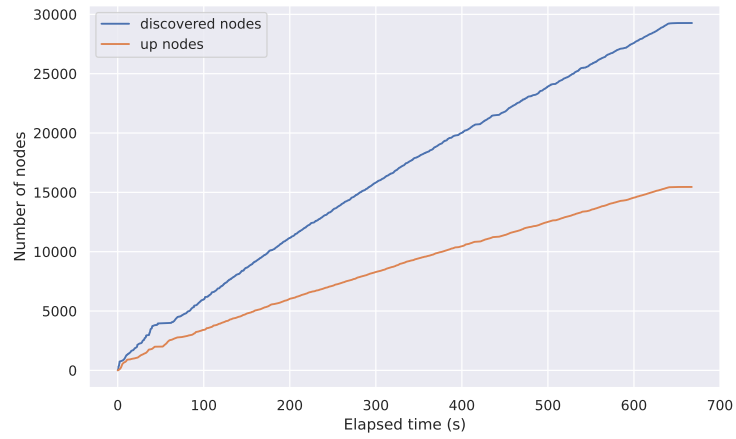


Fig. 3. Cumulative number of discovered/up nodes during a crawl

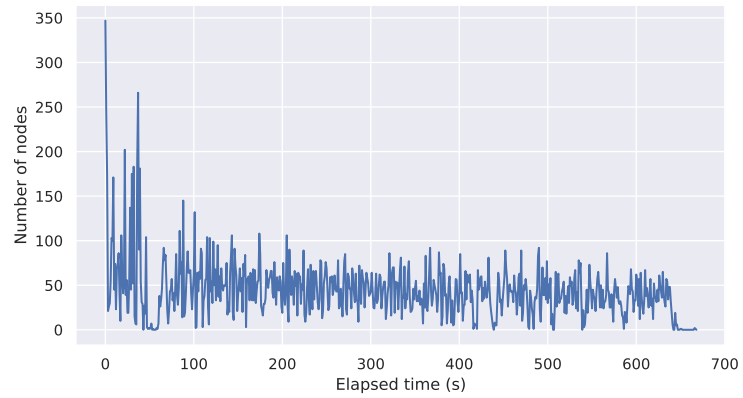


Fig. 4. Number of discovered nodes per second during a crawl

External Validation To assess the quality of our crawler, we wanted to compare its coverage with other available tools. Because the other tools are undocumented, this result is not a proof but rather an insight. This comparison is summarized in Table 1. There are two websites, ethernodes.org [25] and etherscan.io [26], that propose statistics on the Ethereum network. To the best of our knowledge, they both use their own tool that is not open source and their methodology is not documented. In September 2021, Ethernodes and Etherscan could find a similar number of publicly reachable nodes : 3,700 for the first one and 3,000 for the latter. In February and March 2022, Ethernodes could find approximately 6,000 nodes and Etherscan could find 2,500 nodes. In comparison, Crawleth could find in average more than 16,000 publicly reachable nodes and could discover approximately 30,000 nodes in total (reachable and, unreachable) in September 2021. During the sudden rise in the number of discovered nodes in February 2022, Crawleth could find in average more than 33,000 publicly reachable nodes with a peak of 42,000 nodes, and could discover more 50,000 nodes (reachable and, unreachable) during a single crawl. It is important to note that both websites presented a significant drop of their numbers since March/April 2021. Before, Ethernodes and Etherscan counted approximately 10,000 nodes. We can make the hypothesis that they changed the way they count the nodes, for instance by dropping the nodes that are not part of the Ethereum Mainnet, but we cannot be sure because of the lack of official documentation.

Table 1. Comparison of the P2P network size found by Crawleth and other tools

	Date	Average size of network
Crawleth	February–March, 2022	33,000
Ethernodes [25]	February–March, 2022	6,000
Etherscan [26]	February–March, 2022	2,500
Crawleth	September, 2021	16,600
Ethernodes	September, 2021	3,700
Etherscan	September, 2021	3,000
Ethernodes	February, 2021	12,000
Etherscan	February, 2021	8,000
Gao et al. [11]	January, 2019	11,000
NodeFinder [10]	April, 2018	15,454

We can also compare the coverage of Crawleth with the works of Kim et al. [10] on NodeFinder and Gao et al. [11]. In April, 2018 NodeFinder could

see around 15,500 active nodes and in 2019 Gao et al. reported 11,000 active routable nodes. Even though, there are several years apart the crawls, the numbers observed are of the same order of magnitude if we consider the period before February 2022.

Based on the internal and external validation, the methodology of Crawleth can reasonably be considered sound and effective in finding all the nodes composing the Ethereum P2P network.

5 Ethereum’s Network Analysis

This section analyses the data gathered by our crawler during two significant periods: the month of September 2021 and between February 15th and March 7th, 2022. September 2021 is representative of the state of the network since the deployment of our crawler. And in February 2022, we observed a sudden and massive increase in the number of the nodes that deserves a particular focus. Our analyses consider network characteristics that can impact the reliability. The two datasets are publicly released [27] with the sole limitation that IP addresses had to be pseudo-anonymized following ICANN recommendations [28] that offers an acceptable compromise between utility and privacy. More precisely, a salted hash is provided and the last byte of the IP address is set to 0. The scripts used to perform the analysis are also available in our git repository [24], making the whole study fully reproducible.

5.1 Number of Nodes and Distribution

The first metric that we can derive from our datasets is the size of the network and the geolocalization of the nodes. As we can see in Figure 5, the size of the network was quite stable for 7 months (between July 2021 and January 2022) since the deployment of the crawler with a slow but regular grow increasing the number of nodes from 16,000 to 22,000 over the period. Then, the number of nodes rapidly rose on February 23rd 2022 to an ephemeral maximum around 42,000 nodes and declined after to reach a new plateau at 32,000 nodes by early March. Some notable numerical results can also be found in Table 1.

In total, when also including the unreachable nodes, Crawleth discovered approximately 30,000 nodes per crawl in September 2021 compared to the 16,000 reachable nodes. It is also important to note that the crawler could find in total 439,561 Node IDs associated to 103,332 unique IP-Port associations over the month of measurement. The size of this network is very similar to Bitcoin but in comparison to other P2P networks in the literature, especially file-sharing P2P networks like Gnutella, KAD or Bittorent that can easily gather hundreds of thousands of nodes, it remains quite small.

Also, we found that the network was quite well-balanced throughout the world in September 2021, despite a little more nodes being located in Europe than North America or Asia as we can see in Figures 6. However, the peers that joined the network in February 2022 altered the balance (see Figure 7): the

network is now even more concentrated in Europe at the expense of Asia. Please note that the location of nodes is not necessarily correlated with the location of the mining power. It has been demonstrated that disparities exist in this regard in Bitcoin [29] and very similar behaviors can be expected for Ethereum. Unfortunately, the lack of study estimating the geolocalization of Ethereum’s mining pools prevent us to draw a clear conclusion on this aspect.

Generally, a well-balanced geolocalization of the nodes is a good property of the network, as its goal is to rapidly propagate the transactions and blocks, a balanced geographical distribution is more resilient to localized issues. But to further evaluate nodes concentration, we also analyzed their distribution at the network level in Autonomous Systems (AS). In September, 2021 60% of the IP addresses running a node were hosted in only 10 ASes out of 2257 (0.44%). In February and March 2022, it is even more centralized as 71% of the IP addresses were hosted in 10 ASes out of 2292 (0.44%). 9 of these 10 ASes are the same over the two periods and only one of them is not a cloud provider.

To conclude, even though the geographical distribution of the nodes is well-balanced, we observed that most of the nodes are hosted in a very few number of ASes. It denotes a centralization of the network that is not a good property for its resilience. Moreover, the massive number of peers that joined in February 2022 significantly accentuated both geographical and network concentration.

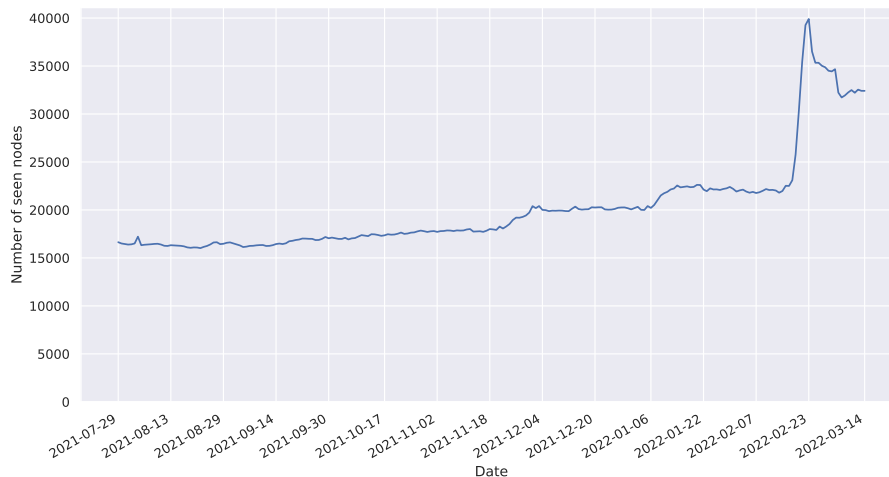


Fig. 5. Number of reachable nodes

5.2 Churn Rate

An interesting metric of the network to analyze is the connection and disconnection rates of the publicly-reachable nodes. It denotes the stability of the network.

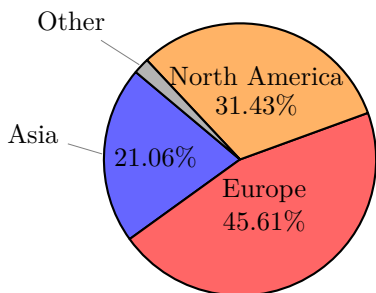


Fig. 6. Geolocation of publicly reachable nodes in Sept. 2021

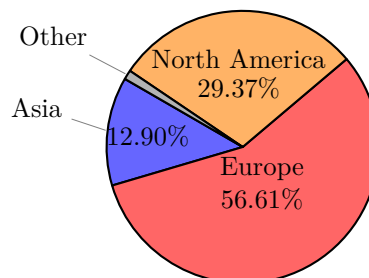


Fig. 7. Geolocation of publicly reachable nodes in Feb.–Mar. 2022

A crawl lasts for approximately 15 minutes, so to observe the hourly churn we aggregated 4 or 5 crawls to get a snapshot of the network over each hour. For the sake of readability, Figure 8 only depicts the hourly churn rate from February 27th, 2022 to March 2nd, 2022 but the tendency is the same throughout the whole measurement period. The disconnection and (re)connection rates per hour and per day are quite constant and stable around respectively 4% and 18%. This is considered low by P2P networks standards which means that the network can be considered stable. Even if a small part of the nodes tend to disconnect often, the size of the network does not change as the nodes reconnect or are replaced by others.

Also, please note that our crawler does not differentiate between the different blockchains using Ethereum’s Node Discovery Protocol. In particular, Ethereum testing instance “Test network” is probably less stable than the main network. Other blockchain systems built on top of the Ethereum Node discovery protocol and less popular than Ethereum may be also less stable.

5.3 Inventory of suspicious nodes

The reliability of P2P networks is mainly based on the proper distribution of the workload on many independent peers. That is why any group of nodes that seems to tamper with the homogeneous distribution of the workload among peers might be dangerous, whether it be an abnormal usage of the network, a wrong configuration, or an intentional Sybil attack. Whatever the root cause, our goal in this section is to make the inventory of nodes that seem to concentrate too much weight in the network and therefore might be considered suspicious.

We consider three categories of suspicious nodes that could be linked because they share a common IP address or a same subnetwork or very close Node IDs. More precisely, we want to identify in our datasets :

- a sub-network (/24) containing more nodes than 10% of its size;
- an IP address carrying more than two identities (Node IDs defined by the public keys);

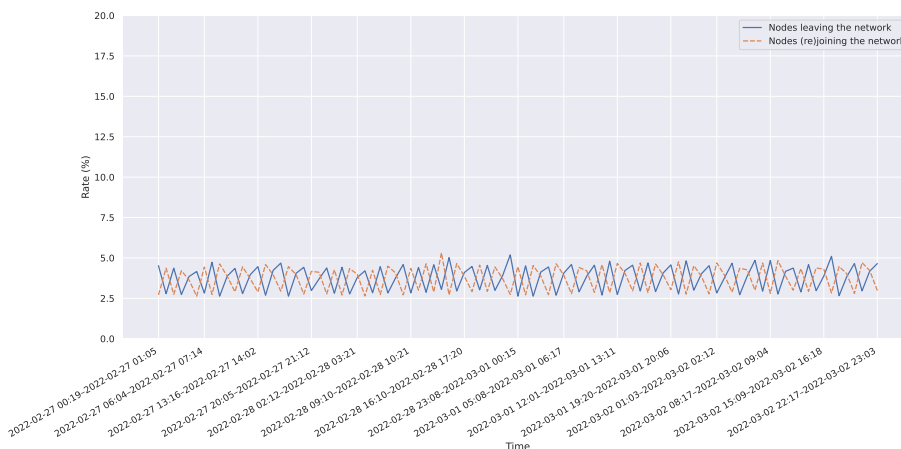


Fig. 8. Churn rate per hour of Ethereum reachable nodes

- Node IDs statistically too close in the distributed hash table compared to the theoretical uniform distribution.

IP addresses concentration in subnets: A few /24 sub-networks contain many IP addresses that are running a node. We set the detection threshold to 10% of the size of a /24 network. In other words, we assume that finding more than 25 nodes in a single sub-network is quite suspicious. Over the whole month of September, 2021 we found five /24 sub-networks containing a total of 214 unique nodes. In average our crawler could find 64 nodes per hour belonging to these few populated subnets. However, those small numbers are not particularly worrisome. Similar numbers were observed in February 2022.

IP addresses holding several identities: We noticed that there are few nodes that hold many Node IDs (i.e. public key). We decided to monitor the nodes that hold more than two identities. For the period of crawl in September 2021, this monitoring unveiled 7,780 unique IP addresses holding 396,963 unique Node IDs. The nodes that forged the highest number of identities, held more than 10,000 Node IDs (maximum ~ 14000). In average our crawler could find 1,217 nodes (21,942 Node IDs) per hour matching this criterion.

During the peak of nodes in February 2022, this behavior was exacerbated: 9,500 unique IP addresses held 183,731 unique Node IDs and 13% of the IP addresses held 75% of the Node IDs. 69.2% of these aliases are held in the top 10 ASes that host most of the IP addresses of the network (see Section 5.1) and are located in the USA or Europe. To compare, the crawler could detect 2,486 deviant nodes per hour during this period.

The concentration of aliases in a very few IP addresses is a typical sign of Sybil attack scenario where an attacker forges many identities and take advan-

tage of them. However, we cannot state on the nature (malicious or simply a misconfiguration) of these nodes. For example, some of these sybil nodes could have been deployed by a blockchain node infrastructure service³. If this assumption proves to be correct, these services should be careful not to exacerbate the centralization of the P2P network.

Non-uniform distribution of the Node IDs in the DHT: Similarly to the previous kind of attack, an attacker could carry a more subtle Sybil attack by intentionally placing a few peers in close proximity to a target identity in the DHT in order to attract requests. For the whole month of September, in Figure 9 we can see that the distribution of size of the shared prefix length between two direct neighbours in the DHT is well distributed, despite irregularities at the end that are not very significant. To confirm, we show in Figure 10 the deviation between measured and theoretical data, represented by the equation that gives the mean number of peers sharing a common prefix, given the total number of peers in the network: $\frac{N}{2^x}$. Our crawler could find 439,561 unique Node ID, so $N = 439561$ here. We can see that there is no obvious deviation between neighbours in the DHT. We observed the same tendency in our dataset from February 2022. Nevertheless, it is possible that a few small localized attacks can occur but would not be visible in statistics calculated at the network level. Typically, a group of 5 peers sharing a common prefix of 22 bits would be statistically unlikely without affecting the whole distribution. At least, we can conclude that no very localized attack (i.e. neighbours sharing a very high prefix) is going on, nor any massive localized attack that would skew the distribution. This can be explained by the fact that no data is actually stored in Ethereum’s DHT, so that such attacks have currently no point.

In conclusion, on the one hand, Ethereum’s DHT shows good properties like a decent geographical distribution, a very low churn, no periodical size variation, a negligible amount of /24 subnetworks holding many peers and no obvious intentional placement of peers near an ID. On the other hand, most nodes are concentrated on a few big ASes, some nodes holds several identities in the network, up to thousands of them for the biggest, and we witnessed a sudden and major increase of the number of peers. We were not able to explain this sudden increase, for example, by correlating these massive arrivals with an external event in the blockchain community. Hopefully, the newcomers did not impact any of the three metrics pointing out suspicious nodes. Nevertheless, this kind of massive event combined with the possibility for a node to create many identities advocate for a permanent monitoring of the P2P network and mechanisms to prevent large scale Sybil attacks.

6 Ethereum Sybil Attack Prevention

Like all fully distributed public P2P networks, that is to say without a central authority that controls access to the network, the P2P network of Ethereum is

³ We can cite some popular services such as Infura or Alchemy.

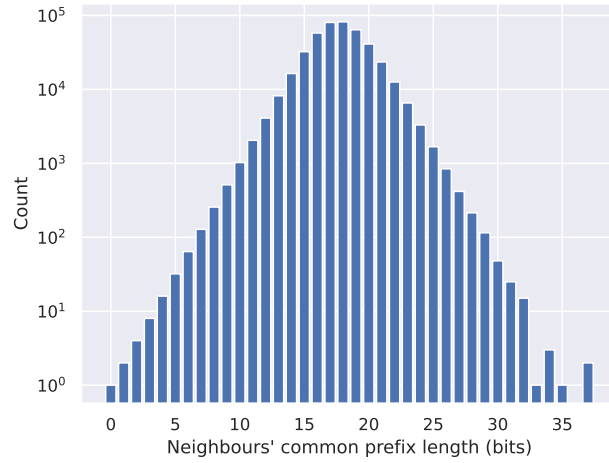


Fig. 9. Distribution of common prefix between neighbours in the DHT

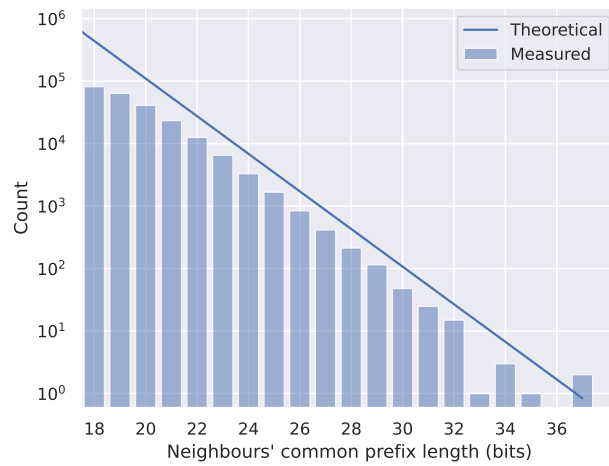


Fig. 10. Theoretical and measured number of peers with a common prefix in the DHT

vulnerable to Sybil attacks, as demonstrated in the previous section with the numerous nodes we found holding several identities. An attacker could easily undermine the P2P network by creating a large number of identities at almost no cost to gain a disproportional influence in the network.

In this context we designed a monitoring system that can 1) detect suspicious nodes globally, 2) advertise them publicly on the blockchain, and 3) performs the revocation in a fully distributed way at the client-side. We will present this system in the following subsections.

6.1 Attacker Model and Attack Scenarios

We assume that all participants, including attackers, are able to create new NodeID (by generating key pair) at will. These new identities do not cost anything and cannot be linked to previous identities.

An adversary could forge multiple identities in the network to carry a Sybil attack. This attack could lead to delays in blocks or transactions propagation and thus favor a mining pool controlled by the adversary [30]. It could also leverage (D)DoS, majority attacks [31], mixing protocol attacks [32]. By monitoring the nodes in the network, our system could detect suspicious peers that constitute a threat of Sybil attack and gossip the information about these peers.

The adversary could also partition the network (eclipse attacks) to reduce the cost and maximize the gain of this attack [33,22]. If our monitoring nodes is included in the partition, it would easily detect the partition by monitoring the difficulty of the new blocks [34]: the difficulty would highly decrease over time in this case.

6.2 Architecture

Our architecture preventing Sybil attacks is composed of three parts, as depicted in Figure 11. The first part (left) consists in the monitoring of the whole Ethereum P2P network and the detection of suspicious nodes. The monitoring is performed by one or several trusted servers running our open-source crawler *Crawleth*. This is necessary to gain a complete view of the network. A fully distributed monitoring performed only by the peers based on their routing table is limited to a partial view that is only precise locally around the peer. This is not sufficient to detect an attack spreading over the entire DHT address space. Once a crawl is performed, the information about the nodes that carry multiple identities is available. The detection of suspicious nodes is made by applying the thresholds defining the three categories presented in Section 6. The second part (center) is the gossiping of these nodes information to all the peers of the network thanks to our smart contract detailed in the next section. After its execution, all the nodes in the network that listen to these events are notified through the event logs of the new suspicious nodes. The third and last part of our architecture preventing Sybil attacks involves each node of the P2P network (right). They must verify the received information through a few targeted *FINDNODE* requests in

order to witness the threat detected by the crawler. This verification step is essential to fully maintain the distributed nature of the P2P network and alleviate the role of the crawler as a simple pointer toward suspicious nodes without any authority on the nodes that remain fully autonomous. Once verified, every node then revokes suspicious nodes by cutting the possible connections they have with them and prevents future ones by constituting a black list. This last part can be either integrated directly in the client or use an external tool we developed and that is presented in Section 6.4.

Previous works proposing defensive mechanisms against Sybil Attacks were either centralized, relying on a central authority to validate peers joining the network [35], or fully distributed, based on a local detection/revocation of suspicious peers [21] but missing an exhaustive view of the DHT to correlate Sybils present in different places. The originality of our approach is to bring the best from both worlds: a global and precise view of the network achieved by a trusted crawler and a fully distributed revocation mechanism to avoid any risk of abuse. We rely on the efficient dissemination of blocks to all the peers to share the crawler knowledge, what was not easily achievable in previous DHTs.

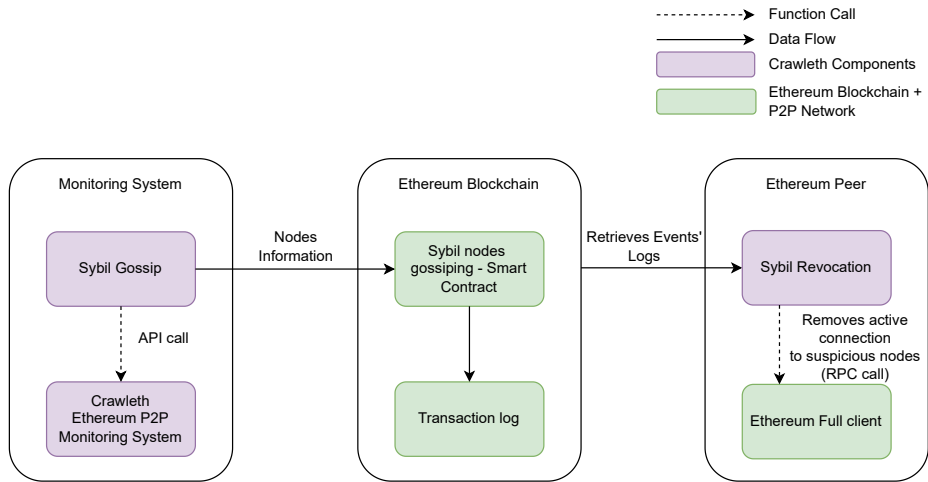


Fig. 11. Sybil attack monitoring system architecture

6.3 Smart Contract

We designed the smart contract presented in the Listing 1 to use event logs to distribute the information about suspicious nodes. In the Ethereum blockchain, every block has a `logsBloom` field which allow any user to search for a specific transaction log (along with the indexed fields from the log). These logs are publicly accessible and can be regenerated from the transaction holding it. The

event in our smart contract publishes the enode (NodeID@IP:Port) of the suspicious nodes but in a structured way to 1) save space in the smart contract by factorizing the information common to a group of suspicious peers, 2) make the verification process easier. Each group of suspicious peers has the data stored differently according to the threat for which they have been identified (labeled as “Subnet”, “Groups” and “Aliases” in Listing 1). Similarly, each threat has a specific verification procedure. While a single FINDNODE request on the right Node ID can possibly find all the suspicious peers that are too close to each others in the DHT address space, a request per suspicious node must be carried out until the threshold is reached defining the suspicious behavior is reached in the case of a suspicious subnetwork or IP address running multiple identities. For instance, a suspicious IP address holding 100 identities only requires the verification of two of them by a peer to allow the revocation. To avoid any overloading, our monitoring system can spread over time the advertisement for suspicious nodes in the smart contract if the number of newly discovered is over a threshold. The smart-contract authorizes a list of allowed users (added by the owner) that can publish the result of different instances of the crawler. Other approved instances of the crawler can complete the data if they witness missed suspicious peers or even replace the main one if needed to ensure the service continuity. The fact to have several trusted crawler’s instances contributes to the correctness and resilience of the monitoring system.

But, storing data in the event logs still has a cost (in addition to the transaction baseline cost): 375 **gas** as base cost, 375 additional **gas** per topics in the event and 8 **gas** for each bytes of data included. For a non-anonymous event, the first topic is the event’s signature (name and type of the arguments). All the indexed arguments are treated as additional topics. Topics are used as criteria to search for specific events. For our smart contract, we do not use indexed arguments, so we only have one topic. **Gas** refers to the fee of a transaction in the Ethereum blockchain, **gas** prices are denoted in **Gwei**, which itself is equal to 10^{-9} ETH. The gas price is not fixed, but is oscillating between 100 and 150 gwei at the time of writing. It is used to reward the miners who include the transaction in a block through the mining process. The gas price is used to regulate the number of transactions and to avoid congestion.

For example, using our smart contract to publish the information about one IP address that has forged 1000 different Node IDs, would cost about 6,000,000 gas units. At a gas price of 100 Gwei, it would cost 0.667604 ETH, which is converted, at the time of writing, to approximately \$ 2,700. It is important to note that a basic transaction (simple ether transfer) already costs 21,000 gas, which is converted to \$ 9 (at the same gas price). We consider that this cost is reasonable for the offered service compared to Ethereum global capitalization (300 Billions dollars at the time of writing). Because the service provided is for the common good, it could be ultimately decided by the community to apply a special treatment to avoid any fee.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.11;
3
4 contract SybilGossip {
5     address owner; //Address of the owner of the smart contract
6     mapping(address => bool) allowedAddr; //Allowed to publish
7     event Subnet(bytes[] subnet, bytes[][] last_bytes_port_nodeid);
8     event Groups(bytes[] enodes);
9     event Aliases(bytes[] enodes);
10
11     constructor() {
12         owner = msg.sender;
13     }
14     modifier onlyOwner() {
15         require(msg.sender == owner, "Ownable: caller is not the owner");
16     }
17
18     //Add the given address to the list of allowed addresses
19     function addUser(address _addressToAllow) public onlyOwner {
20         allowedAddr[_addressToAllow] = true;
21     }
22     modifier isAllowed(address _address) {
23         require(allowedAddr[_address], "You need to be allowed");
24     }
25 }
26
27 /*Publish the information of suspicious nodes in the same subnet
28   _subnet: [subnet1, subnet2, subnet3, ...]
29   _last_bytes_port_nodeid: [
30     [byte_port_nodeid1_1, byte_port_nodeid1_2, ...],
31     [byte_port_nodeid2_1, ...],
32     [byte_port_nodeid3_1, byte_port_nodeid3_2, ...], ...
33   ]
34 */
35 function SubnetPublish(
36     bytes[] memory _subnet, bytes[][] memory _last_bytes_port_nodeid
37 ) public isAllowed(msg.sender) {
38     emit Subnet(_subnet, _last_bytes_port_nodeid);
39 }
40
41 /*Publish the information of nodes that are too close in the DHT
42   _enodes: [enodes1, enodes2, enodes3, ...]
43 */
44 function GroupsPublish(bytes[] memory _enodes)
45     public isAllowed(msg.sender) {
46     emit Groups(_enodes);
47 }
48
49 /*Publish the information of nodes that have too many ID (aliases)
50   _enodes: [enodes1, enodes2, enodes3, ...]
51 */
52 function AliasesPublish(bytes[] memory _enodes)
53     public isAllowed(msg.sender) {
54     emit Aliases(_enodes);
55 }
56 }

```

Source Code 1. Sybil nodes announcement Smart-Contract

6.4 Revocation of suspicious nodes

The suspicious nodes discovered during a crawl are exposed by a web API which source code is available in Crawleth repository [24]⁴. We developed an open-source program “Sybil Gossip” to make the bridge between this API and the blockchain by sending these nodes’ information to our smart contract [36]. This way, after each crawl, up to three events are triggered with the information of these nodes. The website exposing the API also proposes live statistics and charts about the P2P network from its continuous crawling, as well as the history of previous crawls (aggregated per hour for the current day, and per day otherwise).

A second publicly available open-source program “Sybil Revocation” has been developed [36] to retrieve the events’ logs, verify the suspicious nodes existence and remove any active connection of a local Ethereum client once verified. It is represented in the right part in Figure 11. Any user that has deployed an Ethereum node can use this program to prevent its node to keep a connection to a suspicious node. It uses the JSON-RPC API (Remote Procedure Call protocol encoded in JSON) of the official Geth client. Specifically, it uses the “Admin.removePeer” call that is undocumented but exists since 2016 [37]. It can remove any active connection (inbound or outbound) to a specific peer. Because there is no RPC yet to enforce the blacklisting of a node, our tool must periodically check if the local Ethereum clients it protects has connections to known suspicious nodes. An implementation directly within an Ethereum client would make the revocation process more efficient by also sanitizing the routing table and by preventing the suspicious contacts to be shared among peers.

To demonstrate the viability of our architecture, we performed the following final experiment on an Ethereum Test Network. We connected an unmodified peer in the network and run our RPC revocation tool alongside. Then, we manually issued connections toward 10 peers known to be detected suspicious in previous crawls because they carry too many aliases (which constitutes the main current threat to the network we have highlighted in Section 5.3). We performed a new crawl and executed the aforementioned smart contract to add the suspicious peers we found in the logs (because this was done on Ethereum Test Network, no fees were charged). Finally, the RPC revocation tool successfully retrieved the list, performed the check and cut the connection to the 10 peers, thus validating the whole process.

To assess the performance of our system, we also measured the time needed at each step of our architecture from the monitoring to the revocation. The results are given in Table 2. A new crawl is performed approximately every 15 minutes (results are hourly aggregated on <http://crawleth.loria.fr:5000/api/latest/deviant/all>), then the results on suspicious nodes are published in the blockchain using the smart-contract. This step only takes one second on the Test Network but can take up to several minutes on the main Ethereum Network⁵. The list of suspicious nodes is then instantly extracted from the event logs and the verification program from “Sybil Revocation” starts. It checks that

⁴ It is also available live on this server: <http://crawleth.loria.fr:5000>.

Table 2. Time taken by each step of our architecture

Operation	Execution Time
New crawl	~15 minutes
Suspicious node publication	depends on avg transaction time: - in our testnet experiment < 1 s - in the mainnet < 5 min
Retrieval of suspicious nodes from event logs	< 1 s
Verification of suspicious nodes	< 5 s
Revocation	< 1 s

the published nodes actually exist and are indeed suspicious by issuing FIND-NODES requests on the aliases found for a node until the detection threshold is verified. This verification process lasts for approximately 3 to 4 seconds per alias. By parallelizing the verification process, it lasts the same time for all the aliases that must be verified. Once the nodes have been verified, the revocation process is instantaneous. From the moment when the information of the new detected suspicious nodes reaches the peers, the whole process is performed in a few seconds. The main limiting factor is the time taken to perform a crawl but as it takes time to insert malicious nodes in the network, the real key is to perform the crawls continuously to sanitize the network. The crawling time could also be reduced by dividing the address space to crawl between several trusted crawlers.

7 Discussion on storage strategy

We want to motivate the use of event logs in our solution. Our goal is to propose a blockchain-based application that is relevant in terms of cost, performance and ease to develop. Wöhrer et al. [38] explored different architecture designs of blockchain-based applications. For example, in case of a system that would require interacting with components outside the blockchain, a hybrid approach is interesting to design: the application could benefit from the event sourcing which refers to the storage of states as a sequence of immutable events that could be replayed anytime. This approach is very interesting for our system: the list of suspicious nodes found by our monitoring system triggers an event from a smart-contract that is immutably added to the trustless event store of the Ethereum blockchain. In addition, Kostamis et al. [39] studied the cost and practicality of different storing methods in Ethereum. The facts that the event logs are immutably stored and can be replayed to retrieve ancient states make

⁵ <https://ethgasstation.info/>

them interesting to use. All operations executed by a smart-contract have a cost in `gas` (which is derived from `ether`) but using event logs is one of the cheapest way to store data in the Ethereum blockchain. They described another way to store data, by using transaction payload. It is cheaper than event logs but it requires to store the hash of the transactions in an external database to retrieve an ancient state. It increases the complexity of the architecture and the external database may not be as immutable as the Ethereum blockchain. For the reasons mentioned above, we use the Ethereum blockchain to store and distribute the information about the suspicious nodes.

One can argue that we could use a system based on InterPlanetary File System (IPFS) to store the list of suspicious nodes instead of a costly smart-contract. This hybrid system is advertised by the work in [40] and used in practice by Rodrigues et al. [41] for their Blockchain Signaling System to prevent Distributed Denial-of-Service Attacks. But, to the best of our knowledge, there is no IPFS storage provider that is free (we can cite two non-free popular ones: FileCoin and Pinata Cloud). One way to control the costs would be to deploy our own IPFS cluster but this would still have a non-negligible cost. Another drawback is that IPFS would act as another Trusted Third Party which security must also be guaranteed. Additionally, we think that it would not be ethical to rely on an external storage when there is an internal approach for the storage in Ethereum and its market capitalization amounts to more than 300 billions of dollars. Since the service offered is for the global network’s good health, it could ultimately be handled by the Ethereum Foundation as an improvement of the protocols.

8 Conclusion

In this article, we first presented our open source crawler for the Ethereum P2P network. We demonstrated its soundness and analyzed two one-month datasets that we also released. The results show that the Ethereum P2P network exhibits good properties like a decent geographical distribution and a moderate churn rate, but also more concerning ones like a centralization of the nodes in a few ASes. The size of the network has been stable for several months but suddenly rose to double the network size at its peak, without obvious explanation nor pattern that could characterize the newcomers. Then, we considered three categories of threats that denote a risk that a group of peers likely controlled by a single entity get too much weight in the network: a given /24 subnetwork running many peers, an IP address holding many identities, or groups of peers that are too close in the DHT address space. We found thousands of suspicious peers, mainly falling in the second category. The state of the art showed that such Sybil attacks can be used as a lever for different objectives. In this work, we proposed a system to notify when a Sybil attack is at risk and that preventive revocation actions are required. It is based on a smart contract storing the suspicious nodes found by a central monitoring system. Then, each peer of the network must verify the reported anomalies on his own and revoke its connections with the suspicious peers, ensuring a fully distributed revocation. We

provided an external tool based on RPC to provide Ethereum clients with these features.

Several directions are planned to pursue this work. In the short run, we will integrate the revocation mechanism directly into the official Geth client. Then, we will implement Ethereum's higher level protocols (RPLx) to gain more knowledge on the discovered peers. In particular, the lower layer of RLPx will allow us to make the distinction between applications relying on Ethereum's Node Discovery Protocol and focus on Ethereum's mainnet while the upper layer (Ethereum wire protocol) will allow us to check suspicious nodes' behavior by analyzing the transactions and blocs they transmit and detect anomalous data. In the long run, now that Ethereum's Kademlia-based DHT is protected against Sybil attacks, we want to make a better use of the DHT wasted potential by storing actual values in it. In particular storing smart contracts' large data within the DHT with a customizable replication factor and a proportional cost could solve the current problem of prohibitive cost of smart contracts' data within the blockchain [39], that prevents more ambitious applications to be designed with smart contracts.

Acknowledgements

We thank Christophe Belleut and Florent Caspar for their initial work on Crawleth as part of an introduction to research in their school curriculum (TELECOM Nancy, France) which we supervised. We also thank Ambroise Sander for further work and improvement of Crawleth as part of his internship in our research group. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

References

1. Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv:1710.09437 [cs] (Jan 2019), <http://arxiv.org/abs/1710.09437>
2. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16. pp. 3–16. ACM Press, Vienna, Austria (2016), <http://dl.acm.org/citation.cfm?doid=2976749.2978341>
3. Nair, P.R., Dorai, D.R.: Evaluation of performance and security of proof of work and proof of stake using blockchain. In: 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV). pp. 279–283 (Feb 2021)
4. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 357–388. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017)
5. Zhang, R., Preneel, B.: Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 175–192 (May 2019)
6. The Ethereum Foundation: Devp2p - ethereum peer-to-peer networking specifications (2021), <https://github.com/ethereum/devp2p>
7. The go-ethereum developers: Ethereum/go-ethereum, <https://github.com/ethereum/go-ethereum>
8. OpenEthereum DAO: Openethereum, <https://github.com/openethereum/openethereum>
9. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Goos, G., Hartmanis, J., van Leeuwen, J., Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems, vol. 2429, pp. 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
10. Kim, S.K., Ma, Z., Murali, S., Mason, J., Miller, A., Bailey, M.: Measuring ethereum network peers. In: Proceedings of the Internet Measurement Conference 2018 on - IMC '18. pp. 91–104. ACM Press, Boston, MA, USA (2018)
11. Gao, Y., Shi, J., Wang, X., Tan, Q., Zhao, C., Yin, Z.: Topology measurement and analysis on ethereum p2p network. In: 2019 IEEE Symposium on Computers and Communications (ISCC). pp. 1–7 (Jun 2019)
12. Maeng, S.H., Essaid, M., Ju, H.T.: Analysis of ethereum network properties and behavior of influential nodes. In: 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). pp. 203–207 (Sep 2020)
13. Li, Z., Xia, W., Cui, M., Fu, P., Gou, G., Xiong, G.: Mining the characteristics of the ethereum p2p network. In: Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure. pp. 20–30. BSCI '20, Association for Computing Machinery, New York, NY, USA (Oct 2020), <https://doi.org/10.1145/3384943.3409418>
14. Wang, T., Zhao, C., Yang, Q., Zhang, S.: Ethna: Analyzing the underlying peer-to-peer network of the ethereum blockchain. arXiv:2010.01373 [cs] (Oct 2020), <http://arxiv.org/abs/2010.01373>, comment: 14 pages, 14 figures
15. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) Peer-to-Peer Systems. pp. 251–260. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2002)

16. Singh, A., Ngan, T.W., Druschel, P., Wallach, D.S.: Eclipse attacks on overlay networks: Threats and defenses. In: Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. pp. 1–12 (Apr 2006)
17. Steiner, M., En-Najjary, T., Biersack, E.W.: Exploiting kad: Possible uses and misuses. SIGCOMM Comput. Commun. Rev. 37(5), 65–70 (Oct 2007), <https://doi.org/10.1145/1290168.1290176>
18. Wang, P., Tyra, J., Chan-Tin, E., Malchow, T., Kune, D.F., Hopper, N., Kim, Y.: Attacking the kad network. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. pp. 1–10. SecureComm '08, Association for Computing Machinery, New York, NY, USA (Sep 2008), <https://doi.org/10.1145/1460877.1460907>
19. Kohnen, M., Leske, M., Rathgeb, E.P.: Conducting and optimizing eclipse attacks in the kad peer-to-peer network. In: Fratta, L., Schulzrinne, H., Takahashi, Y., Spaniol, O. (eds.) NETWORKING 2009. pp. 104–116. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2009)
20. Cholez, T., Chrisment, I., Festor, O.: Monitoring and controlling content access in kad. In: International Conference on Communications - ICC 2010 (May 2010), <https://hal.inria.fr/inria-00490347>
21. Cholez, T., Chrisment, I., Festor, O., Doyen, G.: Detection and mitigation of localized attacks in a widely deployed p2p network. Peer-to-Peer Netw. Appl. 6(2), 155–174 (Jun 2013), <https://doi.org/10.1007/s12083-012-0137-7>
22. Marcus, Y., Heilman, E., Goldberg, S.: Low-resource eclipse attacks on ethereum's peer-to-peer network. IACR Cryptology ePrint Archive 2018, 236 (2018)
23. Xu, G., Guo, B., Su, C., Zheng, X., Liang, K., Wong, D.S., Wang, H.: Am i eclipsed? a smart detector of eclipse attacks for ethereum. Computers & Security 88, 101604 (Jan 2020)
24. Eisenbarth, J.P.: Crawleth, <https://gitlab.inria.fr/jeisenba/Crawleth>
25. Ethernodes: The ethereum network & node explorer, <https://ethernodes.org/>
26. Etherscan: Ethereum node tracker, <http://etherscan.io/nodetracker>
27. Eisenbarth, J.P.: Ethereum p2p network study, dataset overview (2021), <https://concordia-eth-p2p.lhs.loria.fr/>
28. Internet Corporation for Assigned Names and Numbers: Recommendations on anonymization processes for source ip addresses submitted for future analysis (Aug 2018), <https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf>
29. Eisenbarth, J.P., Cholez, T., Perrin, O.: A comprehensive study of the bitcoin p2p network. In: 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS). pp. 105–112 (Sep 2021), <https://hal.inria.fr/hal-03380595>
30. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security, vol. 8437, pp. 436–454. Springer Berlin Heidelberg, Berlin, Heidelberg (2014), https://doi.org/10.1007/978-3-662-45472-5_28
31. Zhang, S., Lee, J.H.: Double-spending with a sybil attack in the bitcoin decentralized network. IEEE Transactions on Industrial Informatics 15(10), 5715–5722 (Oct 2019)
32. Bissias, G., Ozisik, A.P., Levine, B.N., Liberatore, M.: Sybil-resistant mixing for bitcoin. In: Proceedings of the 13th Workshop on Privacy in the Electronic Society. pp. 149–158. WPES '14, Association for Computing Machinery, New York, NY, USA (Nov 2014)

33. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: Proceedings of the 24th USENIX Conference on Security Symposium. pp. 129–144. SEC'15, USENIX Association, Berkeley, CA, USA (2015)
34. Nguyen, H.L., Eisenbarth, J.P., Ignat, C.L., Perrin, O.: Blockchain-based auditing of transparent log servers. In: Kerschbaum, F., Paraboschi, S. (eds.) Data and Applications Security and Privacy XXXII. pp. 21–37. Lecture Notes in Computer Science, Springer International Publishing, Cham (2018)
35. Dinger, J., Hartenstein, H.: Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In: First International Conference on Availability, Reliability and Security (ARES'06). pp. 8 pp.–763 (Apr 2006)
36. Eisenbarth, J.P.: Sybil-prevention, <https://gitlab.inria.fr/jeisenba/sybil-prevention>
37. The go-ethereum developers: Ethereum/go-ethereum, pull request #2740 (firescar96) : Add ability to remove peers via admin interface, <https://github.com/ethereum/go-ethereum/pull/2740>
38. Wöhler, M., Zdun, U., Rinderle-Ma, S.: Architecture design of blockchain-based applications. In: 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS) (Sep 2021)
39. Kostamis, P., Sendros, A., Efraimidis, P.: Exploring ethereum's data stores: A cost and performance comparison. In: 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS). pp. 53–60 (Sep 2021)
40. Poornima Devi, P., Bragadeesh, S.A., Umamakeswari, A.: Secure data management using ipfs and ethereum. In: Balas, V.E., Hassanien, A.E., Chakrabarti, S., Mandal, L. (eds.) Proceedings of International Conference on Computational Intelligence, Data Science and Cloud Computing. pp. 565–578. Lecture Notes on Data Engineering and Communications Technologies, Springer, Singapore (2021)
41. Rodrigues, B., Scheid, E., Killer, C., Franco, M., Stiller, B.: Blockchain signaling system (bloss): Cooperative signaling of distributed denial-of-service attacks. *J Netw Syst Manage* 28(4), 953–989 (Oct 2020), <https://doi.org/10.1007/s10922-020-09559-4>

Jean-Philippe Eisenbarth is a PhD student in Computer Science at University of Lorraine, France, within the RESIST team at LORIA/Inria Nancy Grand-Est under the supervision of Thibault Cholez and Olivier Perrin. He received his engineering degree (equivalent to a MSc) from TELECOM Nancy, a french *grande école* of engineering, in 2016. He focuses his research on the monitoring, security and applications of P2P networks, especially from blockchain systems.

Thibault Cholez is an Associate Professor at University of Lorraine where he works in the Inria RESIST research team on Data Network Monitoring and Analytics, while he does his teaching activities at the TELECOM Nancy engineering school. He got a Master degree in Computer Science in 2007, and a PhD degree in 2011, both from Université Henri-Poincaré, Nancy, France. He worked formerly as a research associate at the University of Technology of Troyes and at the SnT research center in Luxembourg. He is involved in several research projects at both national and European level.

Olivier Perrin is full professor at University of Lorraine, and works at LORIA in the Coast project. His research interests always been related to data and

service management, and include various topics about collaborative systems, services composition, and distributed systems. He has published several articles in international journals and conferences, and he was involved in many international and European projects. His current work deals with trust and privacy in distributed collaborative applications.