A Comprehensive Study of the Bitcoin P2P Network

Jean-Philippe Eisenbarth *, Thibault Cholez *, Olivier Perrin *
*Universite de Lorraine, CNRS, Inria,
LORIA, F-54000 Nancy, France
Email: {jean-philippe.eisenbarth, thibault.cholez, olivier.perrin}@loria.fr

***Abstract*—The Bitcoin peer-to-peer network ensures the consensus between the different nodes responsible for the propagation of the blocks containing the validated bitcoin transactions. The quality and safety of this network are therefore particularly essential. In this work, we present a study of the public nodes that form the backbone of the Bitcoin p2p network. We analyze the results of our measurement campaign that was made following a well-defined and reproducible methodology. In particular we analyze several criteria that can affect the network resilience: distribution and security assessment of the clients' versions, churn, detection of Sybil nodes, dynamicity and popularity of peers. We also investigate the countermeasures deployed to prevent an accurate inference of the network topology and show their effectiveness.**

***Index Terms*—Blockchain, Bitcoin p2p network, network measurement, p2p protocol, network security**

I. Introduction

The cryptocurrencies have drawn a lot of attention in the last decade with Bitcoin being the most popular one. It is based on a decentralized blockchain that is a distributed transaction ledger whose records are append-only and publicly auditable. This ledger is managed by an underlying peer-to-peer (p2p) network that agrees on a common order of the transactions using a distributed consensus. This combination is called a permissionless blockchain.

Due to the popularity and success of Bitcoin, its protocols, network and security have been widely studied [1], [2], [3], [4] and several attack vectors have been revealed [5], [6], [7] [8], [9], [10]. The Bitcoin community has used these studies to develop fixes and countermeasures to these attacks. But, it lacks thorough studies of the network as a whole, and the ensuing implications.

In a previous work [11], we presented our publicly available dataset [12] composed of snapshots of the Bitcoin network and we performed a first analysis of the collected data (number of nodes, geolocalization and clients distribution). We showed that the size of the Bitcoin p2p network is very stable with approximately 7800 reachable nodes, that more than 98% of the peers execute the official client, and that peers composing the network are well balanced throughout the world.

In this paper, we pursue our analysis of the p2p network with new metrics to further assess its resilience. Our dataset is composed of information about the nodes composing the p2p network gathered in daily crawls during one month. The analysis are made by open source scripts available in our git repository (URL in section IV-A) and thus fully reproducible, by opposition to previous studies in the domain. We highlight some metrics that characterize the network. Among these metrics, we analyze the popularity of peers, the inventory of software vulnerabilities that affect clients' versions and their distribution among the deployed nodes as well as the detection of Sybil nodes. We also evaluate the applicability of several link inference techniques from the state of the art on today's clients that now include countermeasures.

The rest of the paper is organized as follows. Section II presents some background on the Bitcoin network and its protocols. Then, Section III surveys the related work of previous studies on this network. In Section IV, we present our crawling strategy to make the dataset, then we analyze it in Section V. Finally, in Section VI we evaluate the countermeasures deployed to prevent the inference of the network topology and Section VII concludes the paper.

II. Background

In this paper, we focus on the implementation of the Bitcoin Core client [13], which is the official implementation, documented in the Bitcoin's community documentation [14]. The other unofficial implementations (btcd [15], bitcoinj [16] or bcoin [17] for example) are very similar but we will not discuss their specific features here.

There are four main types of messages in the Bitcoin's p2p network:

- network discovery messages
- transactions or blocks announcements
- transactions or blocks requests
- transactions or blocks deliveries

We will focus on the network discovery messages (also called overlay network protocol). Now we will describe how the Bitcoin p2p network works [1] [18] [7] [19] [20].

There are two types of nodes in the Bitcoin p2p network: the full nodes and the lightweight nodes. A full node downloads every block and transaction and verifies that they respect all the Bitcoin's consensus rules. If it is not the case, the invalid blocks or transactions are rejected, no matter the network's global state about them. In order to use Bitcoin in a trustless and private way, a user needs to

run a full node wallet. In contrast, a lightweight client only downloads the block headers to verify the authenticity of the transactions. This method is called Simplified Payment Verification (SPV). This type of client relies upon full nodes that become trusted third parties.

Essentially, when a full node starts up, it needs to connect to other active peers of the network to synchronize its local blockchain state with the one the network agreed upon. At the very first startup, a node does not know any other peers of the network. Most likely, it will use the DNS seeders to learn about them. Basically the DNS seeders continuously crawl the network to create a list of active reachable peers. When they receive a DNS request from a new node, they reply with a subset of chosen IP addresses they know. The list of DNS hostnames to query is hardcoded in the Bitcoin client (e.g. seed.bitcoin.sipa.be, dnsseed.bluematt.me, dnsseed.bitcoin.dashjr.org, etc.).

A node has two databases in which it stores IP addresses of other peers of the network: the `new` and the `tried` tables. These two tables are organized into buckets, 1024 buckets for the `new` table and 256 for the `tried` table. Each bucket can contain a maximum of 64 entries. The `new` table contains the addresses of peers to which the node has not yet tried to connect to. The `tried` table contains the addresses of peers to which the node has been able to connect successfully and that are known as reachable.

After the local database has been seeded with some active peers, the node tries to connect to 8 peers by default using unencrypted TCP connections (called outbound connections). Additionally, the node can be configured (port forwarding, access enabled in the firewall, etc.) to accept connections (called inbound connections) from other peers, up to 117 by default. Such a node — called a *listening node* — may issue unsolicited `ADDR` messages to advertise its neighbors that it accepts inbound connections. These neighbors may relay this message to their own neighbors and the distribution may carry on this way by following a gossip protocol. A node could also request explicitly to discover other active peers in the network by sending a `GETADDR` message to its neighbors. To determine how many addresses a node needs to answer, it calculates *m* the minimum between 23% of the number of nodes contained in all the buckets and a maximum number set to 2,500. Then, it responds with `ADDR` messages containing information (ip, port, timestamp) of the *m* peers retrieved randomly from the buckets (a single `ADDR` message can contain up to 1000 buckets entries information). At maximum, a node would send 3 messages, each containing respectively 1000, 1000 and 500 addresses from its tables. This is how the ADDR messages are documented for the Bitcoin Core client but we noticed that it is no longer the case for the versions released after 2014 ($>$ v0.10.0). Since then, a node sends at maximum 1000 addresses in one message. The list of addresses that contains 23% of the number of the nodes is cropped to 1000. The reason is not documented and not clear, even inside the community and they plan to remove this limitation in the future [21].

Also, a node maintains its connections up-to-date by verifying periodically the state of the nodes it is connected to by issuing application-level `PING` messages and waiting for the `PONG` responses. The two types of connection (inbound and outbound) are used to disseminate transaction and block messages (announcement, request and delivery).

III. Related Work

These last few years, the Bitcoin network has drawn the attention of researchers on its technical aspects and its security. In particular, there are several publications on its protocol and network.

Miller et al. [3] proposed a technique called *AddressProbe* to discover the public topology of the Bitcoin network. It consists of issuing `GETADDR` messages to the peers to get a snapshot of the network, and a map of connections can be inferred by analyzing the timestamps. They observed that the Bitcoin graph is well connected but it is not a purely random graph, it contains communities. They discovered few influential nodes that represent approximately 2% of all nodes but they account for 75% of the mining power. If a transaction (respectively a block) reaches these nodes it is likely to be included in a block (respectively to extend the main chain). They hypothesized that these influential nodes may be gateways — a local instance of a Bitcoin node — to mining pools.

Gencer et al. [19] characterized the Bitcoin and Ethereum networks by analyzing measurements gathered by a system they created that is called *Blockchain Measurement System* (BMS). It connects to a peer, collects measurements then disconnects and proceeds to the next peer. It targets Bitcoin IPv4, IPv6, Tor nodes and Ethereum IPv4 nodes. BMS is used to measure the peer-to-peer latency, the provisioned bandwidth and to infer a view of the network. They observed that Bitcoin nodes are more likely hosted in data centers (clustered) and the miners are mostly centralized but the consensus protocols would benefit from a more decentralized mining process.

Imtiaz et al. [4] showed the impact of the churn in the Bitcoin network in 2018. They observed that 97% of the nodes experience churns and that the propagation time is increased in average by 135% compared to nodes not experiencing churn.

Heilman et al. [7] presented an attack on the Bitcoin network that completely isolates a specific node by monopolizing all of its outgoing and incoming connections. Concerning the outgoing connections of a node, it tries to connect to the peers it learned about previously (from the `tried` and `new` tables). The authors managed to populate the tables of the victim with their own malicious nodes and to force the eviction of honest nodes. They showed that in the worst case for the attacker, there is a 85% probability to successfully perform this attack with a 4600-nodes botnet. Since the publication of this attack, the authors and the

Bitcoin developers implemented countermeasures to prevent this specific eclipse attack.

To the best of our knowledge, even if some authors indicate their methodology, either their dataset is not publicly available or the source code of the software used (crawlers, scripts to build the statistics, algorithms...) is not given, preventing reproducible results. None also proved that the crawler used converges and is sound. Beyond methodology questions, some important criteria are yet unknown like nodes popularity in contact lists, the state of the network regarding software vulnerabilities and link inference, or the presence of Sybil nodes, what motivated this work.

IV. Measurement strategy

A. Crawling methodology

From February 7–March 2, 2020 we operated a crawling node on the Bitcoin's main network. We forked and improved the open source crawler used by the bitnodes.io website and we also published our version on Github [22]. We extended this project to add some missing features: keeping track of the nodes discovering rate to validate the methodology as well as harvesting and exporting all the content of the buckets of the nodes. We also added all the functions to calculate the statistics and generate the figures we present in this paper. The methodology of this software involves first a crawler that tries to concurrently establish connections to the peers of the Bitcoin network and second, another program that runs in parallel and that keeps track of the connectivity of the discovered nodes. This program connects to all the nodes returned by the crawler and sends periodically (every 60 second) a PING message to the connected nodes to verify if they are still connected. When a crawl is finished, the nodes that are tagged as reachable by this program are exported with additional information such as geolocalization, hostname, Bitcoin client used, among others.

It is important to note that our infrastructure lacked an IPv6 link and a TOR proxy to test and integrate these nodes. Thus, all the discovered nodes are IPv4 nodes. They accounted for approximately 75% of the network whereas IPv6 and Tor nodes accounted for 10% and 15% of the network respectively. Please note that in the dataset we publicly released, IP addresses are pseudo-anonymized following ICANN recommendations [23] that offers an acceptable compromise between utility and privacy. More precisely a salted hash is provided and the last byte of the address is set to 0.

Also, nodes that are running a Bitcoin protocol version older than 70001 were not taken into account during a crawl because of compatibility issues with prior versions of the protocol. It corresponds to the Bitcoin Core v0.8.0 client and below (before February, 2013). Pappalardo et al. [24] showed that already in 2016 the vast majority of nodes in the network were running a protocol version more recent than 70001. Consequently, we consider that the crawler could reasonably ignore the nodes that are running an older version of the protocol.

B. 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 that established connections to all the IPv4 reachable peers — i.e. listening nodes — of the Bitcoin peer-to-peer network. 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. We also set some network kernel settings as specified in appendix A. The machine operated the crawling node 24/7 for the whole period.

C. Validation

In order to validate the quality and the consistency of our measurements, we want to assess that a crawl ends when all possible peers are discovered. A crawl proceeds as follows:

1) the crawling node is bootstrapped by receiving a small subset of the reachable nodes from known DNS seeders which addresses[1] are hardcoded in the client ;
2) it tries to establish a connection to the newly discovered nodes ;
3) then it sends `GETADDR` messages to all the nodes it is connected to ;
4) the crawling node disconnects from a node once it received the `ADDR` messages in response to its `GETADDR` message ;
5) it loops back to step 2 until the crawling node does not discover new nodes (one it did not process yet) in a certain amount of time.

More precisely, the newly discovered nodes are stored as pending and the crawling node tries to establish a connection to them. These nodes are retrieved from the `GETADDR` sent by the nodes the crawler is connected to. This list of addresses is filtered in order to dismiss stale nodes (lastly seen online more than 8 hours from the current time of crawl) for efficiency purpose (older nodes are not likely to be connected). A crawl stops when there are no pending nodes left (checked every 10 seconds). In practice a crawl did not exceed 3 minutes and the program was set to restart a new crawl every 4 minutes. As shown in Fig. 1, a crawl stops learning new nodes after 2 to 3 minutes and the number of new nodes discovered over time is decreasing after a peak at the beginning.

Also, we deployed five independent ground-truth Bitcoin Core v0.20.1 nodes. We did not modified the source code, they ran the official Bitcoin Core client from Github. They joined the p2p network and four of them had approximately 30 active connections whereas the fifth node had 70 active connections. The crawler was able to find all of them. The methodology of the crawler can reasonably be considered effective in finding all the accessible nodes in the Bitcoin p2p network.

[1] dnsseed.bitcoin.dashjr.org, dnsseed.bluematt.me, seed.bitcoin.sipa.be, seed.bitcoinstats.com, seed.bitcoin.sprovoost.nl and seed.bitnodes.io

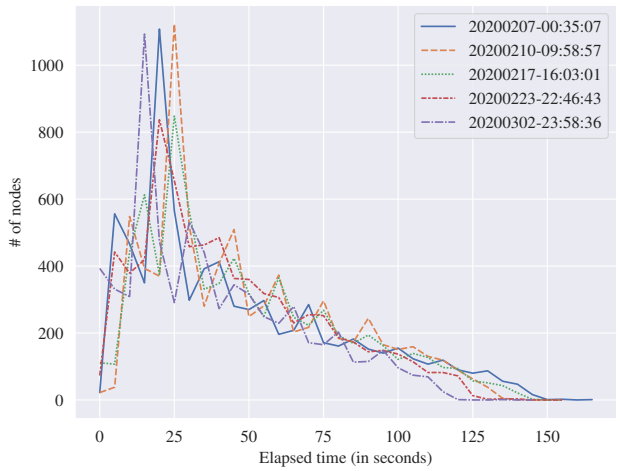

Fig. 1. Number of new reachable nodes discovered throughout five crawls.

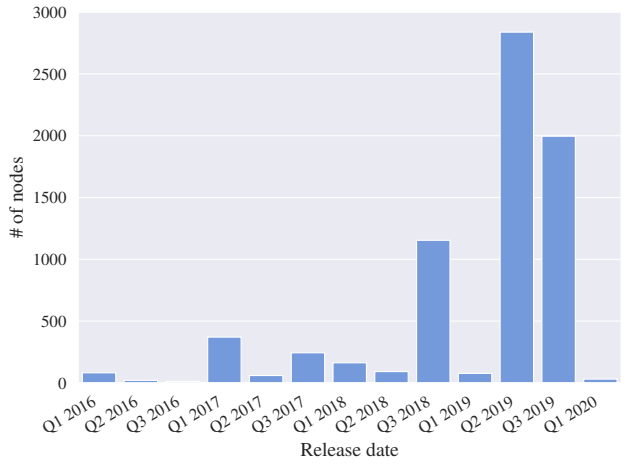

Fig. 2. Temporal distribution of the `Bitcoin Core` versions during a crawl on March 2nd.

V. Bitcoin's Network Analysis

In this section, we analyze several metrics that are derived from the crawls of the Bitcoin network. Usually, in a cyber attack there is a reconnaissance phase in which the attacker gathers a lot of information about the target that can be leveraged to find one or several weaknesses. For instance, a network partitioning attack requires the attacker to highlight nodes of interest to be disconnected from the network. For each aspect of the network, we will consider what it tells regarding the robustness of the network against attacks.

A. Clients' versions distribution and security

To ensure overall security, the nodes in a p2p network have to use the latest version of the clients. To assess this information, we classify the clients according to their official release date [25] to observe how the users keep the client up-to-date. As shown in Fig. 2, a majority of nodes is quite up-to-date by running a version of the client released in the last year. To keep the figure readable, we did not display the temporal distribution of the versions released prior to 2016: it only concerns about a hundred nodes which client version was released between 2013 and 2015. We can see in Figure 2 that a significant amount of clients (more than 30%) is running versions that could be considered as deprecated (more than one year), what may have security implications when some vulnerabilities are exploited.

So, we analyzed some common vulnerabilities and exposures (CVE) that affect the Bitcoin Core client and are listed in Table I. We wanted to understand how the freshness of network clients could help to lower the risks of an overall attack.

The CVE (5) and CVE (6) can be exploited via the same component: the RPC implementation. In the first case it would allow an attacker to inject arbitrary data in the debug log and in the second case it would allow an attacker to steal currencies. The impact of the latter vulnerability is higher and critical but very unlikely to happen since it needs a local access to the node. These two vulnerabilities concern a significant part of the network (more than 20%) and the users run the risk to face these two attacks by not updating their nodes.

TABLE I
Some CVE that affect the Bitcoin Core client in the network at the end of the crawl period (i.e. March 2nd 2020)

	Versions affected	Severity [26] (out of 10 points)	Part of the network that is vulnerable
CVE-2016-10724 (1)	$<$ 0.13.0	7.5	1.62%
CVE-2016-10725 (2)	$<$ 0.13.0	7.5	1.62%
CVE-2017-18350 (3)	$<$ 0.15.1	5.9	9.42%
CVE-2018-17144 (4)	0.14.0 to 0.16.3	7.5	7.25%
CVE-2018-20586 (5)	$<$ 0.17.1	5.3	22.30%
CVE-2018-20587 (6)	0.12.0 to 0.17.1	5.5	21.96%

The CVE (1) and CVE (2) also concern the same component: the alert system. It is a system introduced in the version 0.3.10 and withdrawn in the version 0.13.0 that allowed the holders of a private alert key to broadcast signed messages to all the clients in the network to inform on some important events (such as accidental forks). The CVE (2) allows an attacker to block a special final alert with a non-final alert whereas the CVE (1) allows an attacker to perform a denial of service (by memory exhaustion).

There are 2 other CVE that allow denial of service: CVE (3) and (4). The first one concerns the version of Bitcoin Core prior to 0.15.1 that would experience stack-based buffer overflow if the client uses a SOCKS proxy controlled by the attacker and the latter concerns the version from 0.14.0 to 0.16.3 that would experience an application crash due to duplicate input given by miners.

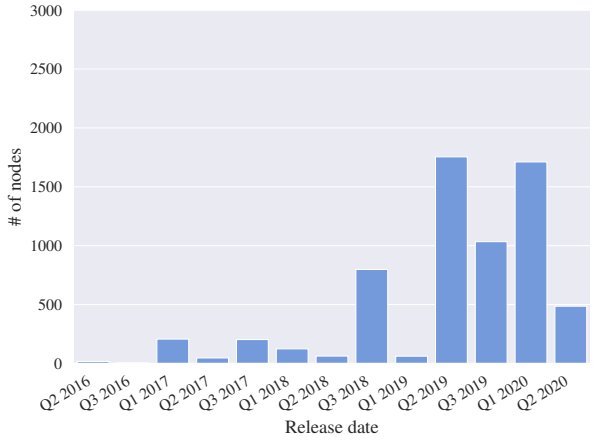

Fig. 3. Temporal distribution of the `Bitcoin Core` versions during a crawl on June 7th.

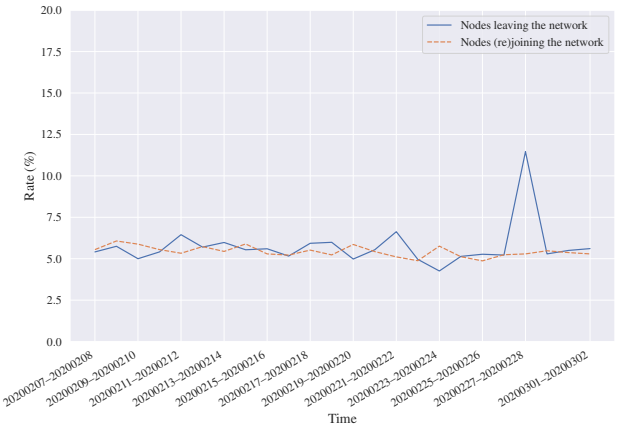

Fig. 4. Percentage of nodes in a one-day window leaving or (re)joining the network.

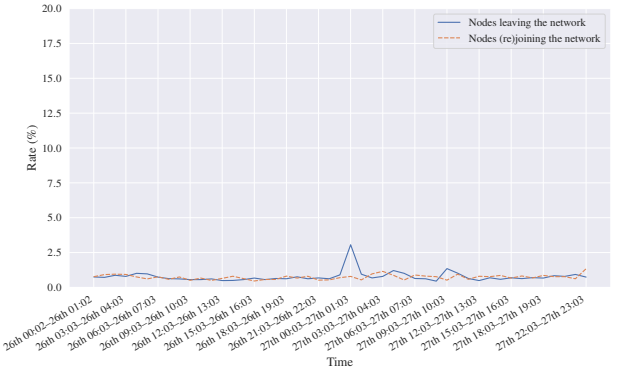

Fig. 5. On February 26th and 27th, the percentage of nodes in an one hour window leaving or (re)joining the network.

We think that the CVE that allow denial of service are the most concerning ones for the network. They concern approximately 10% of the network, but they could be leveraged by an overall attack which purpose is to lower the majority, as described in [7] and [8]. For the network as a whole to be secured, this implies that all users must keep their client up-to-date. The relative freshness of the clients shows that this consideration is not particularly worrisome regarding the Bitcoin network.

Also, an interesting property of the network to analyze is the reactivity of the peers to update their client when a new version is released. In Fig. 3 we show the temporal distribution of the clients a few days straight after the `0.20.0` version release (early June, 2020). We can see that there is a significant amount of early adopters that did upgrade their clients fastly. Also, since March (see Fig. 2), there is an important number of clients that have upgraded their clients to a more recent version (i.e. there is a shift from Q1 2019 to Q1 2020). However, we can still notice that there is an important number of users that keeps using deprecated versions of the client released before Q2 2019. For releases before 2016, the trend is equivalent to the trend of Fig. 2. A little effort from the community to incite the upgrade of deprecated versions could be beneficial for the security of the network as a whole, even if it is not currently worrisome as we stated before.

B. Churn

We want to know if the churn is low and stable.We monitored the disconnections and reconnections of the nodes for the whole measurement period. It means that we observed the nodes that were leaving or joining the network from one day to the next one. As shown in Fig.4, the rate of nodes leaving or joining the network is quite stable between 5% and 6% per day. Concerning the dashed curve, we did not distinguish the new nodes that were joining and the nodes that were rejoining the network after having shortly left it. We can see that there is a peak of disconnections on February 27th. We decided to monitor the churn from one hour to the next one on February 26th and 27th. As shown in Fig. 5 the peak occurred on February 27th between 0:03 (UTC+1) and 13:03 (UTC+1) with the maximum number of disconnection occurring between 0:03 (UTC+1) and 1:03 (UTC+1). In our dataset, we did not observe any reconnection rate afterward that would cancel out this disconnection rate.

Despite this peak, we can note that the network of reachable nodes is stable even if almost all nodes experience churn at some point as shown by Imtiaz et al. [4].

C. Distribution of peers' popularity in the contacts' list

We were then interested in the content of the responses of these `GETADDR` messages. Particularly, we wanted to know how often a given IP address appears in the buckets of the reachable nodes. As before, we aggregated one hour of crawls to have more general results. We can see in Fig. 6 the occurrences of the IP addresses, sorted in ascending order. It is interesting to observe the trend of the curve: a vast majority of IP addresses that only appear in the buckets of a few reachable nodes, and some IP addresses that tend

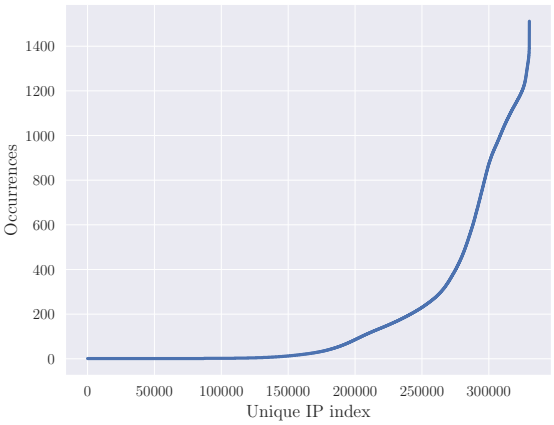

Fig. 6. Number of occurrences of a node's IP address in the GETADDR responses gathered during one hour of crawls

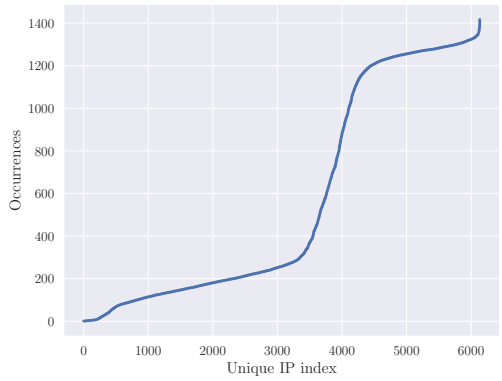

Fig. 7. Number of occurrences of a reachable node's IP address in the GETADDR responses

to appear frequently in the buckets of a large number of reachable nodes. It is important to note that all these IP addresses are not necessarily reachable nodes, they could also be offline nodes (short-term or long-term disconnection) or unreachable nodes (behind a NAT, Bitcoin wallets, etc.). We observed that almost all the first 150,000 IP addresses (the least frequent ones) are unreachable nodes (i.e. our crawler did not establish a successful connection). Whereas, in the 1,000 most frequent ones, approximately 80% are reachable nodes. The trend is important: it seems that an IP address that is returned in a GETADDR request by a significant number of nodes is more likely to be an active node in the Bitcoin network. We can see in Fig. 7 the occurrences of the reachable nodes only. The curve shows two clear inflection points, that allows to exhibit 3 main categories of nodes: the first 50% that are the least frequent, the next 25% shows an average but non homogeneous popularity and the last 25% are very popular.

We can deduct that the network tends to propagate the IP addresses of reachable nodes. On the opposite, the exponential grow at the end of the curve shows that nodes' popularity is vastly unbalanced: only a fraction of all the retrieved IP addresses are highly popular while p2p networks are more resilient when they are well balanced. Those very popular nodes could be targets of choice in case of attack.

D. Sybil attack detection

Several studies concerning Sybil attacks [27] on the Bitcoin P2P network [2], [28] show that an attacker could leverage this type of attack to weaken the consensus (disruption of the blocks/transactions propagation, (D)DoS attack, ...). A simple technique to detect if such an attack is ongoing is to verify if there is a significant number of nodes that has the same IP address or that is running in the same subnetwork. In our dataset, we found that there is only four /24 submask in which 10 or less nodes are running. Furthermore, the /24 submask in which the highest number of nodes are running only carries 19 nodes. Regarding the size of the network (≈8000 nodes) this is not significant. It seems that during the period covered by our crawls there is no suspicion about an ongoing Sybil attack. In order to assess the security of the network, one could monitor the whole network by running permanent crawls and using this technique to unveil a possible ongoing Sybil attack.

VI. Link inference

A. Previous attempts

Knowing about the actual links between peers is of prime importance to progress on our understanding of the Bitcoin p2p network. There has been several works on topology inference but some have been made partly obsolete by countermeasures introduced in the source code to avoid topology discovery [21], [29].

In 2014, Biryukov et al. [2] presented a Sybil attack against the p2p network whose purpose is the deanonymization of the Bitcoin users. They leveraged the fact that their tool was connected to all the nodes of the network and could inject false peers' information and follow its path and thus infer the topology. This methodology is not suitable for our work since it is an active attack that greatly disturbs the network.

In 2019, Delgado-Segura et al. [30] proposed a technique to infer the topology by leveraging orphan transactions and the mempool structure. The idea is to send 3 special transactions (a parent, a flood and a marker transactions) to different nodes and check if the marker transaction can be found in another node by sending an INV message. The main issues about this technique are the cost and the impact on the network. They evaluated to 400$ (back in 2019) the cost of the execution of this technique to infer the topology of the network at a given time. They also admitted that they did not execute this technique on the Bitcoin's mainnet because of the potential negative impact it could have. For these two reasons, this technique is not suitable for our work.

In 2018, Deshpande et al. [20] and in 2020, Essaid et al. [31] presented a similar technique to infer the topology. They developped an emulator and emulated a probable topology

of the network but not the real one. We do not know how likely this topology is, how the Bitcoin emulator works and how close to a real Bitcoin client it is.

In 2015, Miller et al. [3] presented their methodology to infer links between peers and deduct the topology. Their technique leverages the fact that for outgoing connections, a node updates the timestamp associated with the connected node each time it receives a message from this peer. Thus, they crawled the network to discover all the nodes of the network and retrieved the timestamps of their contacts' list. Then they verified if the associated timestamps are recent (less than two hours) and inferred the connections between the nodes. It was a well-thought technique but the Bitcoin developers decided to implement a countermeasure [29] against this technique.

B. Evaluation of the feasibility of the timestamp technique despite the countermeasures

The Bitcoin Core developers decided to remove the update of the timestamp when a node receives messages on an active connection [29]. The updates of the timestamp for others criteria are kept (for example on the initiation of a connection between two nodes)

We wanted to evaluate this countermeasure and be sure that the topology could not be inferred anymore, thus we conducted some experiments. The first steps consists in crawling each peer's contact list by sending many `GETADDR` messages. Because each answer will contain random contacts, it is not possible to discover all of them but we can still get an accurate view by multiplying requests. Figure 8 shows the number of new contacts learned from our five ground-truth nodes according to the number of requests made. 180 requests seem sufficient to discover most contacts in a list. Considering the delay between messages, most of the contacts of a node can be discovered in approximately two hours. This could be reduced by using a distributed approach, but the low churn rate of the network highlighted in Section V-B does not support the need for faster crawls.

Then we wanted to assess the possibility to infer a peer's links by observing the timestamps reported for each contact. We retrieved the contact lists of our ground-truth nodes and analyzed them. We noticed that the timestamps update are not recent enough to use the Miller et al. technique. We analyzed them further to try to unveil a pattern in the update frequency that could be used to infer the topology.

In a temporal window of 24 hours, we noticed that the timestamp are still updated but the number of updates seems arbitrary. We can see in Table II that the false positive rate is very high (more than 99%) and even the false negative rate is unacceptable (50%) when we consider too many updates.

To conclude, it seems that given the countermeasures implemented in the Bitcoin client it is very complicated to infer the exact topology of the network nowadays. The Bitcoin developers even decided to make it more difficult to infer the topology using the responses to `GETADDR` messages by caching these responses [21] (included in August 2020).

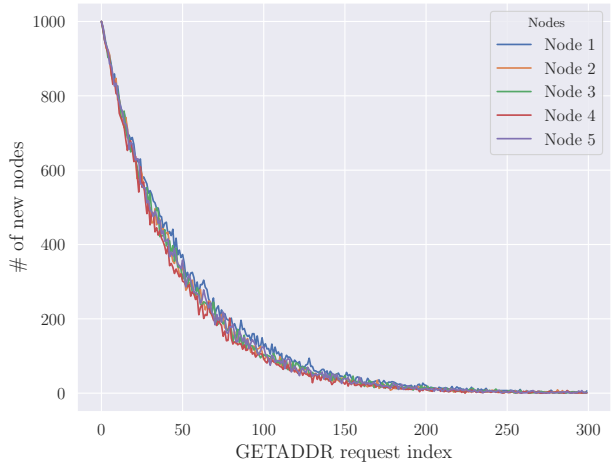

Fig. 8. Number of new nodes discovered during `GETADDR` requests

TABLE II
NUMBER OF CONNECTIONS INFERRED GIVEN THE NUMBER OF TIMESTAMPS' UPDATES WITHIN 24 HOURS

	3 updates	7 updates	20 updates
True positives	10	9	5
False positives	9318	7039	2968
True negatives	44878	47157	51228
False negatives	0	1	5

VII. CONCLUSION AND FUTURE WORK

We presented an analysis of the Bitcoin's public nodes p2p network and its characterization. Based on in-vivo measurements over one month, we showed that there is little churn in the network, no Sybil attack, and that recent countermeasure are effective. But it exhibits more concerning properties like the fact that a significant part of the network tends to update the client version slowly, as well as the very unbalanced popularity of peers, even among the reachable ones.

Nevertheless, it is important to note that the analysis we showed in this paper concerns the public nodes (namely reachable full nodes) in the Bitcoin p2p network and it cannot be trivially generalized to all the nodes in the network (wallets, lightweight nodes, miners…). Yet, these public nodes are crucial, since they act as a backbone of the complete Bitcoin network: wallets and lightweight clients need the public full nodes to connect to the network and retrieve the blocks and transactions.

In our future work, we plan to study a probable network topology and emulate it in an instrumentalized environment to evaluate the resiliency of the network against large scale attacks or failures and help to improve it.

Acknowledgement
This project has received funding from the European Union's

Horizon 2020 research and innovation programme under grant agreement No 830927 .

REFERENCES

[1] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*. Trento, Italy: IEEE, Sep. 2013, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/document/6688704/

[2] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of Clients in Bitcoin P2P Network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, Nov. 2014, pp. 15–29. [Online]. Available: https://doi.org/10.1145/2660267.2660379

[3] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes," 2015.

[4] 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)*. Seoul, Korea (South): IEEE, May 2019, pp. 431–439. [Online]. Available: https://ieeexplore.ieee.org/document/8751297/

[5] I. Eyal and E. G. Sirer, "Majority Is Not Enough: Bitcoin Mining Is Vulnerable," in *Financial Cryptography and Data Security*, N. Christin and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, vol. 8437, pp. 436–454. [Online]. Available: http://link.springer.com/10.1007/978-3-662-45472-5_28

[6] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore, "Game-Theoretic Analysis of DDoS Attacks Against Bitcoin Mining Pools," in *Financial Cryptography and Data Security*, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, vol. 8438, pp. 72–86. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44774-1_6

[7] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 129–144.

[8] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. Saarbrucken: IEEE, Mar. 2016, pp. 305–320. [Online]. Available: http://ieeexplore.ieee.org/document/7467362/

[9] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. San Jose, CA, USA: IEEE, May 2017, pp. 375–392. [Online]. Available: http://ieeexplore.ieee.org/document/7958588/

[10] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 496–511.

[11] J.-P. Eisenbarth, T. Cholez, and O. Perrin, "An open measurement dataset on the Bitcoin P2P Network," in *2021 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2021, [Soon to appear].

[12] J.-P. Eisenbarth, "Bitcoin P2P network study Dataset Overview," 2020. [Online]. Available: https://concordia-btc-p2p.lhs.loria.fr/index.html

[13] The Bitcoin Core developers, "Github bitcoin/bitcoin," Bitcoin, [Accessed on March 20th 2020]. [Online]. Available: https://github.com/bitcoin/bitcoin

[14] The Bitcoin community, "Satoshi Client Node Discovery," [Accessed on March 10th 2020]. [Online]. Available: https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery

[15] The btcsuite developers, "Github btcsuite/btcd," Bitcoin in Go, [Accessed on March 20th 2020]. [Online]. Available: https://github.com/btcsuite/btcd

[16] The bitcoinj developers, "Github bitcoinj/bitcoinj," bitcoinj, [Accessed on March 20th 2020]. [Online]. Available: https://github.com/bitcoinj/bitcoinj

[17] The bcoin developers, "Github bcoin-org/bcoin," [Accessed on April 2020]. [Online]. Available: https://github.com/bcoin-org/bcoin

[18] S. Feld, M. Schönfeld, and M. Werner, "Analyzing the Deployment of Bitcoin's P2P Network under an AS-level Perspective," *Procedia Computer Science*, vol. 32, pp. 1121–1126, 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S187705091400742X

[19] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum Networks," in *Financial Cryptography and Data Security*, S. Meiklejohn and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, vol. 10957, pp. 439–457. [Online]. Available: http://link.springer.com/10.1007/978-3-662-58387-6_24

[20] V. Deshpande, H. Badis, and L. George, "BTCmap: Mapping Bitcoin Peer-to-Peer Network Topology," in *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. Toulouse: IEEE, Sep. 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8548904/

[21] The Bitcoin community, "Cache responses to GETADDR to prevent topology leaks by naumenkogs · Pull Request #18991 · bitcoin/bitcoin." [Online]. Available: https://github.com/bitcoin/bitcoin/pull/18991

[22] J.-P. Eisenbarth, "Github jpeisenbarth/bitnodes forked from ayeowch/bitnodes." [Online]. Available: https://github.com/jpeisenbarth/bitnodes/tree/add_statistics

[23] Internet Corporation for Assigned Names and Numbers, "Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis," Aug. 2018. [Online]. Available: https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf

[24] G. Pappalardo, T. Di Matteo, G. Caldarelli, and T. Aste, "Blockchain inefficiency in the Bitcoin peers network," *EPJ Data Science*, vol. 7, no. 1, p. 30, Dec. 2018. [Online]. Available: https://epjdatascience.springeropen.com/articles/10.1140/epjds/s13688-018-0159-3

[25] The Bitcoin Core developers, "Releases of bitcoin/bitcoin." [Online]. Available: https://github.com/bitcoin/bitcoin/releases

[26] NIST, "NVD - CVSS v3 Calculator." [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

[27] J. R. Douceur, "The Sybil Attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer, 2002, pp. 251–260.

[28] P. Swathi, C. Modi, and D. Patel, "Preventing Sybil Attack in Blockchain using Distributed Behavior Monitoring of Miners," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Jul. 2019, pp. 1–6.

[29] The Bitcoin Core developers, "Reduce fingerprinting through timestamps in 'addr' messages. · bitcoin/bitcoin@9c27379." [Online]. Available: https://github.com/bitcoin/bitcoin/commit/9c2737901b5203f267d21d728019d64b46f1d9f3

[30] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds. Cham: Springer International Publishing, 2019, pp. 550–566.

[31] M. Essaid, S. Park, and H.-T. Ju, "Bitcoin's dynamic peer-to-peer topology," *International Journal of Network Management*, vol. 30, no. 5, Sep. 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2106

APPENDIX A
CRAWLER'S KERNEL SETTINGS

```
net.core.rmem_default=33554432
net.core.wmem_default=33554432
net.core.rmem_max=33554432
net.core.wmem_max=33554432
net.ipv4.tcp_rmem=10240 87380 33554432
net.ipv4.tcp_wmem=10240 87380 33554432
net.ipv4.ip_local_port_range=2000 65500
```