Root Cause Analyses for the Deteriorating Bitcoin Network Synchronization

Muhammad Saad‡, Songqing Chen†, and David Mohaisen‡
‡University of Central Florida †George Mason University
saad.ucf@knights.ucf.edu, sqchen@gmu.edu, mohaisen@ucf.edu

***Abstract*—The Bitcoin network synchronization is crucial for its security against partitioning attacks. From 2014 to 2018, the Bitcoin network size has increased, while the percentage of synchronized nodes has decreased due to block propagation delay, which increases with the network size. However, in the last few months, the network synchronization has deteriorated despite a constant network size. The change in the synchronization pattern suggests that the network size is not the only factor in place, necessitating a root cause analysis of network synchronization.**

In this paper, we perform a root cause analysis to study four factors that affect network synchronization: the unreachable nodes, the addressing protocol, the information relaying protocol, and the network churn. Our study reveals that the *unreachable* nodes size is 24x the *reachable* network size. We also found that the network addressing protocol does not distinguish between *reachable* and *unreachable* nodes, leading to inefficiencies due to attempts to connect with *unreachable* nodes/addresses. We note that the outcome of this behavior is a low success rate of the outgoing connections, which reduces the average outdegree. Through measurements, we found malicious nodes that exploit this opportunity to flood the network with *unreachable* addresses.

We also discovered that Bitcoin follows a round-robin relaying mechanism that adds a small delay in block propagation. Finally, we observe a high churn in the Bitcoin network where ≈8% nodes leave the network every day. In the last few months the churn among synchronized nodes has doubled, which is likely the most dominant factor in decreasing network synchronization. Consolidating our insights, we propose improvements in Bitcoin Core to increase network synchronization.

I. Introduction

Synchronization in the Bitcoin network ensures that all nodes have an up-to-date and consistent view of the blockchain, whereas the lack of such a view, i.e., nodes being behind the current state of the blockchain, makes the Bitcoin network vulnerable to the partitioning attacks [22]. In Bitcoin, a peer-to-peer (P2P) network of more than 10,000 nodes [13], synchronization is affected by the block propagation delay. Recent work has shown that the synchronization in the Bitcoin network deteriorated in 2018 [22], when compared to 2013 [5], due to the increasing network size: between 2013 and 2018, the number of *reachable* nodes increased from ≈3K nodes to ≈10K nodes, thereby increasing the block propagation delay.

The fundamental reason why the block propagation delay increases as a function of the Bitcoin network size is twofold. First, in the current design of the Bitcoin protocol, each *reachable* node in the Bitcoin network can connect to only eight other *reachable* nodes, making that the outdegree of each node in the network. As a result, increasing the network size while fixing the outdegree will only mean that longer paths will connect any two nodes in the network on average, and a block has to traverse more hops to reach its destination, eventually negatively affecting the Bitcoin network synchronization by increasing the block propagation delay. Second, an increasing number of Bitcoin nodes are possibly hosted across different Autonomous Systems (ASes) [6], [1], with various hosting patterns. Given the spatial node distribution across those ASes, and the complexity (length and policy) of the routes between those ASes, the increase of the number of nodes will also increase the block propagation delay.

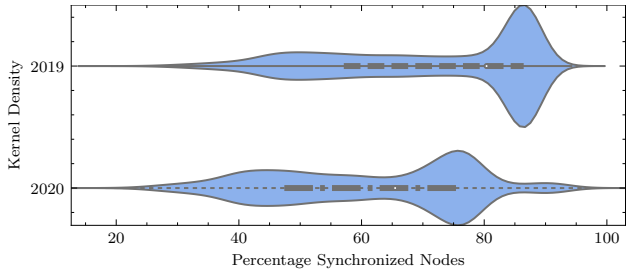


Figure 1. Bitcoin network synchronization in 2019 and 2020. Synchronization is determined by the percentage of nodes with the up-to-date blockchain. In 2019, the mean and median network synchronization were 72.02% and 80.38%, respectively. In 2020, the mean and median network synchronization decreased to 61.91% and 65.47%, respectively. The kernel density shape also shows that the Bitcoin network synchronization decreased in 2020.

Given the stability of the Internet's latency distribution, one would anticipate the overall Bitcoin network synchronization to stay the same with the absence of any significant change to the Bitcoin protocol, and given a constant network size. Surprisingly, however, recent measurements show that, while the *reachable* network size has remained constant (≈10K nodes), the network synchronization has deteriorated even further, as shown in Figure 1, which captures a kernel density plot of the network synchronization in 2019 and 2020.

To investigate what caused such changes, in both cases, we collected the Bitcoin network data from Bitnodes [2] and studied the synchronization. Through analysis, we found that the number of *reachable* nodes between September and December 2019 was ≈10K, where the average synchronized nodes' percentage was ≈72%. However, we found that while the average number of *reachable* nodes stayed the same between January and April 2020, at ≈10K, the average percentage of synchronized nodes decreased to only ≈62%.

The decreasing network synchronization is alarming, suggesting that the Bitcoin network is becoming more vulnerable to partitioning [22]. Moreover, these results suggest that the network synchronization cannot be only attributed to the network size, and there must be other hidden factors affecting it, which warrants further exploration through a root cause analysis. To this end, in this paper, we explore four plausible

factors that may influence the block propagation and network synchronization, which we briefly discuss in the following.

Unreachable Network. There are two types of Bitcoin nodes, *reachable* and *unreachable*. The *reachable* nodes establish outgoing connections with and accept incoming connections from other nodes. The *unreachable* nodes (often behind a NAT [6]) only establish outgoing connections. In the prior work, block propagation was measured by observing the interactions among *reachable* nodes [2], [22], [18], while the growth of *unreachable* nodes and their impact on network synchronization is not well understood yet. While the *reachable* network size remains unchanged, as shown in Figure 1, the *unreachable* network size might be growing and may influence the block propagation. Therefore, the first part of our analysis includes a mapping of the *unreachable* network, aiming to determine its impact on the Bitcoin network synchronization.

Addressing Protocols. When outbound connections of a node are dropped, the node tries to establish new connections with other nodes until the outbound slots are filled. While making those connections, the node does not distinguish between *reachable* and *unreachable* IP addresses. Ideally, a node's IP address database, `addrMan`, should contain only *reachable* addresses so that each outgoing connection is successful. In contrast, if `addrMan` is dominated by *unreachable* IP addresses, the node wastes time in failed connection attempts and, as a result, might not immediately receive a block if the outgoing slots are not filled. The second part of our analysis is to study the Bitcoin addressing protocol and empirically evaluate the success rate of outgoing connection establishment. We conjecture a low success ratio of outgoing connection establishment due to the addressing protocol will contribute to the deteriorating network synchronization.

Relaying Protocols. The Bitcoin ideal design [15], [7] assumes that each node "broadcasts" blocks to the entire network in a *lock-step* synchronous [7], [21] manner, meaning that the block is concurrently released to all connections. Exploring how the Bitcoin Core implements the broadcast mechanisms is essential, although not done before. Therefore, the third part of our analysis is to study the Bitcoin block relaying protocol and analyze its impact on network synchronization.

Network Churn. The Bitcoin network is *permissionless* and nodes can leave the network at any time. The network churn decreases the average node outdegree and network synchronization. Moreover, when new nodes replace the synchronized nodes, it could take the new nodes several days to download the Bitcoin blockchain. Only after downloading the blockchain and synchronizing with the network, they will be able to propagate newly mined blocks. Therefore, a high churn is undesirable. The fourth part of our analysis is to measure the network churn and how it affects synchronization.

Contributions and Roadmap. We pursue a measurement-based approach for evaluating these factors. First, we set up a data collection system that connects to all *reachable* nodes and collects IP addresses of *unreachable* nodes. Our longitudinal analysis captures the number of *reachable* and *unreachable* nodes, and characterizes the *reachable* network churn. Our source code inspection unveils characteristics of the block relaying protocol and the network addressing protocol. We conduct experiments to evaluate those factors' impact on block propagation. Our key findings and contributions are as follows:

1) We conduct a comprehensive mapping of the Bitcoin network. As a result, we discover ≈29K *reachable* and ≈694K *unreachable* nodes, respectively. By probing the ≈694K *unreachable* nodes, we find that ≈54K *unreachable* nodes are active at any time (§IV-A).
2) We conduct a Bitcoin Core source code analysis. We identify characteristics and unveil a major limitation in the Bitcoin addressing protocol. We note that the addressing protocol does not distinguish *reachable* and *unreachable* nodes, thus increasing the failure rate of the outgoing connections. Our experiments on a custom Bitcoin node show a failure rate of 88.8% (§IV-B).
3) Through various measurements, we analyze and detect malicious peers behavior by inspecting the `ADDR` message payload. We identify 73 *reachable* nodes that exploit weaknesses in the addressing protocol to flood the Bitcoin network with *unreachable* IP addresses (§IV-B).
4) We study the *in situ* block relaying protocol in Bitcoin and measure its impact on block propagation. We find that Bitcoin implements a round-robin block relaying for each connection, rather than the simultaneous block broadcast. Our experiments on a custom node show that the round-robin relaying can add up to 17 seconds of delay in relaying block to the last connection (§IV-C).
5) We measure the impact of the network churn on synchronization. Our analysis reveals that *reachable* nodes have a short life. Each day, ≈8% *reachable* nodes leave the network, replaced by an equal number of new *reachable* nodes. Compared to 2019, the churn among the synchronized *reachable* nodes has doubled in 2020 (§IV-D).
6) Capitalizing on our measurements and analysis, we propose refinements to the current Bitcoin Core design to improve network synchronization (§V).

Additionally, the paper includes the related work in §II and concluding remarks in §VI.

II. Related Work

In this section, we discuss the prior related work on Bitcoin network mapping and the network synchronization.

Network Mapping. In 2013, Decker *et al.* [5] conducted the first measurement study on the Bitcoin network to estimate the number of *reachable* and *unreachable* nodes, and the block propagation among the *reachable* nodes. They reported that the Bitcoin network consisted of ≈3K *reachable* and ≈16K *unreachable* nodes. In 2017 and 2018, two measurement studies [1], [22] used the Bitnodes dataset to report the number of *reachable* nodes in the Bitcoin network. In [1], the authors reported ≈8K *reachable* nodes, and in [22], the authors reported ≈13K *reachable* nodes in the Bitcoin network. Concurrent to these works, Neudecker *et al.* [17] also deployed crawlers in the Bitcoin network to estimate the number of *reachable* nodes. Their findings are consistent with Bitnodes, establishing a consensus in the research community on the number *reachable* nodes in the Bitcoin network.

While independent measurements on the number of *reachable* nodes have been consistent, we note various discrepancies in the number of *unreachable* nodes reported in prior works. For instance, in 2019, Naumenko *et al.* [16] reported that there were ≈60K *reachable* and *unreachable* nodes in the Bitcoin network. Given the consistent reports on the 10K *reachable*

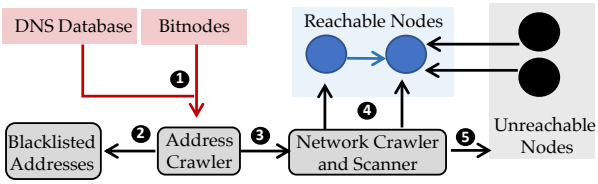


Figure 2. The data collection workflow. The "Address Crawler" collects IP addresses of *reachable* nodes from the DNS server database and Bitnodes, removes blacklisted IP addresses, and forwards them to the "Network Crawler and Scanner", which operates our Bitcoin node that sends `GETADDR` messages to the *reachable* nodes. After collecting IP addresses of *unreachable* nodes, it sends them a `VER` message using *Scapy*. *Unreachable* nodes that respond to the `VER` message are labeled as *responsive* nodes.

nodes [2], the number of *unreachable* nodes reported by [16] is ≈50K. However, their results are significantly different from another study conduced by Park *et al.* [19], reporting ≈1M *unreachable* nodes in the Bitcoin network. Our findings are closer to the results reported by Park *et al.* [19], since we discovered ≈694K *unreachable* nodes in the Bitcoin network.

A closely related question to the network size is the distribution of nodes across ASes, which motivates the partitioning attack models presented in [22], [1], [10], [23]. We note that all prior works only considered *reachable* nodes in the spatial partitioning attacks. For instance, in [22], the authors claimed that if 8 ASes are hijacked, ≈50% Bitcoin nodes can be isolated. In practice, this claim is only valid for *reachable* nodes which, as we show in §III, are significantly fewer than the *unreachable* nodes. As such, a more practical spatial partitioning attack must acknowledge the *unreachable* nodes, since those nodes can also maintain a blockchain, exchange transactions, and mine blocks. We use our results in §III to refine the partitioning attack models presented in prior works.

Synchronization. The *reachable* nodes contribute to the synchronization in the Bitcoin network by relaying blocks to other *reachable* and *unreachable* nodes. Therefore, information propagation among the *reachable* nodes is critical to determine the network synchronization. In 2013, Decker *et al.* [5] observed that a block would reach 90% *reachable* nodes within 12 seconds. They also used the propagation characteristics to amortize the cost of the majority attacks from 51% to 49% hash rate. In 2018, Saad *et al.* [22] used Bitnodes dataset to observe the block propagation and the network synchronization. Compared to [5], their results showed an increasing block propagation delay and decreasing network synchronization. They reported instances where only 30% *reachable* nodes received a block in 10 minutes. They attributed the decreasing network synchronization to the increasing number of *reachable* nodes which, as discussed earlier, does not hold in the current network.

III. Data Collection Methodology and Overview

Our analysis is based on the data we have collected from the Bitcoin network in a duration of 60 days (04 April, 2020 to 04 June, 2020). For this purpose, we have implemented our data collection system as shown in Figure 2. In this section, we present our data collection methodology, step-by-step, and an overview of our dataset.

A. Collecting Reachable Bitcoin Node Addresses

As a first step of our data collection, we collect all reachable addresses in the Bitcoin network. Since our data collection system relies on some key designs of the Bitcoin network protocol, we first present some of such details before discussing our data collection methodology.

Default Connection Limits. By default, in Bitcoin, a *reachable* node can establish 8 outgoing and 117 incoming connections, while an *unreachable* node can only establish 8 outgoing connections [3]. Since *unreachable* nodes drop incoming connections, no two *unreachable* nodes can directly connect to each other. As such, the information exchange (*i.e.,* transactions or blocks) between two *unreachable* nodes is enabled by *reachable* nodes.

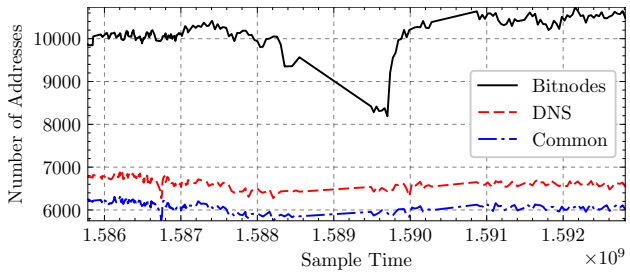
Node Bootstrapping. When a node joins the Bitcoin network for the first time, it queries nine DNS seeders that are hard-coded in the *chainparams.cpp* file [9]. The DNS seeders return a list of IP addresses to which the node may establish outgoing connections. Once a connection is established, the node sends a `GETADDR` request to each connected node and receives an `ADDR` message in response. The `ADDR` message contains up to 1000 IP addresses that the sender selects from its `addrMan` database. If the `addrMan` database has less than 1000 IP addresses, the sending node sends all those addresses in the `ADDR` message. In each `ADDR` message, a node also sends its own IP address with the current UNIX timestamp.

Upon receiving the `ADDR` message, a node stores IP addresses in either a `tried` table or a `new` table. The `tried` table stores addresses that the node has connected to in the past, while the `new` table stores addresses that the node sees for the first time. To establish a new outgoing connection, the node randomly selects an IP address from the `new` or `tried` table with an equal probability. If a successful connection is established to an IP address from the `new` table, that IP address is moved to the `tried` table.

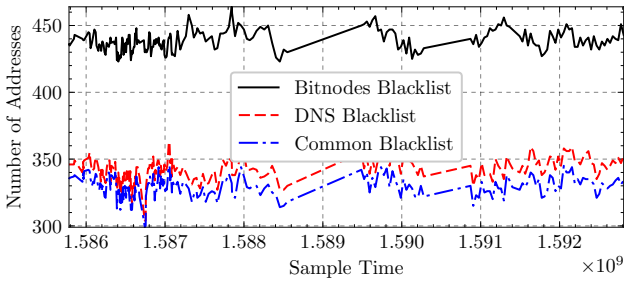
Design of our Collection System. Given these characteristics of a Bitcoin node, we set up our data collection system to connect to the *reachable* nodes and discover the *unreachable* nodes. Our system is shown in Figure 2, where we first collected the IP addresses of *reachable* nodes from Bitnodes and a DNS server database. Bitnodes has been extensively used in prior works [22], [1] to sample *reachable* nodes, and we adopt the same approach in our study. Additionally, we gained access to a Bitcoin DNS server database maintained by Luke Dashjr who has hard-coded his DNS server address in Bitcoin Core *chainparams.cpp* file [12]. Luke Dashjr's DNS database records IP addresses of nodes that queried his DNS server. The objective of using the DNS database is to connect with the *reachable* addresses that may be skipped by Bitnodes, thus ensuring full coverage as much as possible.

Ethical Considerations. During this study, we were advised to avoid connecting to any Bitcoin node hosted in the national critical infrastructure sector (*i.e.,* military infrastructure). In compliance, we compiled a list of 4 million IP addresses that belong to the critical infrastructure. As shown in Figure 2, after collecting IP addresses from Bitnodes and the DNS server database, we removed the IP addresses that are mapped to the critical infrastructure sector and excluded them from analysis.

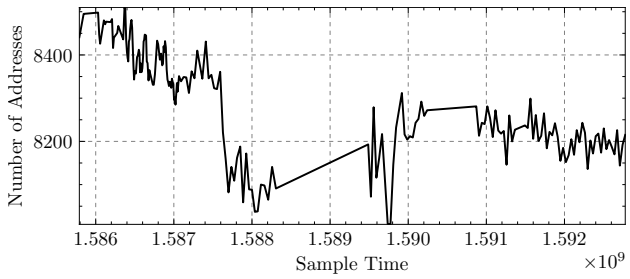
After removing these addresses, we provide the remaining set of *reachable* IP addresses to the "Network Crawler and



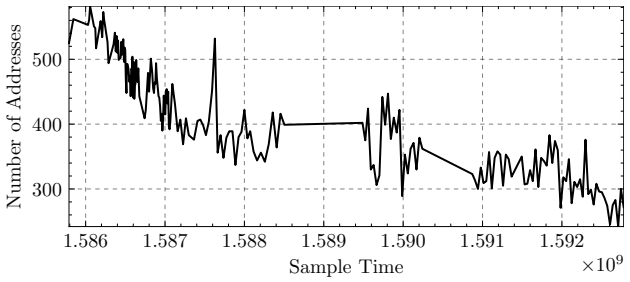
(a) IP addresses of Bitnodes, DNS, and common among two.



(b) Banned IP addresses of Bitnodes, DNS, and common among two.



(c) All *reachable* nodes to which we connected.



(d) *Reachable* nodes not in Bitnodes

Figure 3. Results obtained from preliminary experiments. On average, from Bitnodes and DNS server database, we collected 10,114 and 6,637 IP addresses, respectively. Among them, 439 and 342 addresses belonged to the critical infrastructure, respectively. Our Bitcoin node connected with 8,270 nodes on average.

Scanner" (Figure 2) that consists of a Bitcoin node equipped with a packet generation tool called *Scapy*. The Bitcoin node connects with all the *reachable* nodes and exchanges `ADDR` messages to collect IP addresses of *unreachable* nodes. It then uses *Scapy* to probe the *unreachable* nodes and mark the ones that responds to the `VER` message.

Overview of Reachable Addresses. In Figure 3, we report (1) the number of IP addresses collected from Bitnodes and the DNS server Figure 3(a), (2) the number of excluded IP addresses in Bitnodes and the DNS server Figure 3(b), (3) the number *reachable* nodes to which we connected Figure 3(c), and (4) the number of *reachable* nodes, skipped by Bitnodes Figure 3(d). On average, Bitnodes provided 10,114 IP addresses, among which 439 were excluded. The DNS server database provided 6,637 IP addresses, among which 342 were excluded. 6,078 IP addresses were common in both Bitnodes and the DNS database. Among them, 329 were excluded.

In 60 days, our Bitcoin node connected with 28,781 unique *reachable* IP addresses. In each experiment, our node connected with 8270 *reachable* nodes on average, where 95.78% of all nodes used the default 8333 port while the remaining used 264 other unique ports. Moreover, some *reachable* IP addresses provided by the DNS server database were not present in Bitnodes. As shown in Figure 3(d), on average, we connected with 404 IP addresses that were not present in the Bitnodes dataset. This shows that using the DNS server database for experiments does extending our coverage of the *reachable* network.

B. Collecting Unreachable Addresses

After connecting to the *reachable* nodes, our node sent `GETADDR` requests to all reachable nodes. In response, the nodes replied with `ADDR` message containing up to 1000 IP addresses selected from their `new` and `tried` tables [9]. In algorithm 1, we provide the methodology of collecting *unreachable* addresses from the *reachable* nodes. For simplicity,

Algorithm 1: Discovering *unreachable* IP addresses.

Input: *reachable* IP addresses in N_r
Initialize Empty list of *unreachable* IP addresses N_u
foreach $P_i \in N_r$ **do**
  Send `GETADDR` and **Receive** `ADDR` messages
  if ip addresses $\in$ `ADDR` $\notin N_r$ or N_u **then**
    foreach ip address $\notin N_r$ **do**
      $N_u \leftarrow$ **ip address**
    repeat
  else
    foreach ip address $\notin N_r$ **do**
      $N_u \leftarrow$ **ip address**
    continue
Output: IP addresses of *unreachable* nodes in N_u

we define N_r and N_u as two IP address sets for *reachable* and *unreachable* nodes, respectively. We further define P_i as an address in N_r to which our Bitcoin node sent `GETADDR` request. If the `ADDR` message from P_i contains one unique address that was not sent in prior `ADDR` messages, our node repeats the `GETADDR` request. If a new message contains all IP addresses that were sent in previous `ADDR` message, we stop sending `GETADDR` messages and assume that the node has sent all addresses from its tables. Through iterative requests, we collected IP addresses from each node's `new` and `tried` tables. For all addresses received, our node filtered *reachable* addresses from Bitnodes and the DNS server database to obtain the *unreachable* addresses.

C. Discovering Responsive Unreachable Addresses

Once the network crawler and scanner (Figure 2) obtain the list of all *unreachable* addresses, it scans them to detect *responsive* nodes. In this paper, the term *responsive* nodes refer to *unreachable* nodes that drop the incoming connection by

Algorithm 2: Discovering *responsive* addresses in N_u.

Input: *reachable* IP addresses in N_u
Initialize List of *responsive* nodes N_3
foreach $P_i \in N_u$ **do**
  Send VER message
  if P_i responds to VER **then**
    $N_3 \leftarrow P_i$
  else
    mark P_i as *silent*
Output: IP addresses *responsive* nodes in N_3

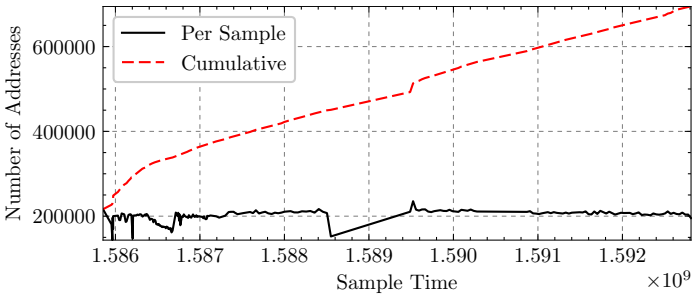


Figure 4. Longitudinal analysis of *unreachable* addresses collected from the network. The black line shows the unique IP addresses collected in each experiment and the red line shows the cumulative number of unique IP addresses collected in 60 days. The gap between the two lines shows that in each experiment, new IP addresses appeared in the network. Overall, we collected ≈694K unique IP addresses of *unreachable* nodes.

responding to the VER message. As a result, despite the node being *unreachable*, we know that it is running Bitcoin.

To verify our method, we deployed three *unreachable* nodes inside our network and sent a VER message through our *reachable* node outside of our network. We observed that all three *unreachable* nodes dropped our connection by sending a response to the VER message with the FIN flag set to 1. We applied this methodology to all the *unreachable* IP addresses received from the *reachable* nodes. If the *unreachable* address responded to VER message with FIN set to 1, the address is marked as *responsive*. algorithm 2 outlines our methodology of detecting *responsive* nodes.

Considering the high volume of *unreachable* addresses, we manually crafted Bitcoin VER message in *Scapy* and applied algorithm 2. The script was deployed on a commodity computer that sent 250 parallel requests to the *unreachable* addresses. Note that the procedure of using VER message is heuristic since it may only work for nodes that allow such incoming requests through their firewalls. It is possible that a node is *unreachable* and running Bitcoin while it has blocked all incoming requests. In that case, our heuristic will not be able to detect the *responsive* node. Therefore, the number of *responsive* nodes that we have detected provides a lower bound estimate of *unreachable* nodes in the network.

IV. Analysis and Results

In this section, we present our analysis and main findings. For each of the four factors outlined in §I, we evaluate their impact on block propagation and network synchronization.

A. Unreachable Nodes

In 60 days, we collected 694,696 unique *unreachable* IP addresses, with ≈195K addresses in each experiment. Among those addresses, 615,083 (88.54%) used the default 8333 port while 79,613 (11.46%) used 9,414 unique ports. Figure 4 presents the *unreachable* addresses obtained in each experiment and the cumulative number of unique *unreachable* addresses collected in all experiments. Among the total of 694,696 addresses, 163,496 (23.54%) were *responsive*, and ≈54K (27.69%) addresses were *responsive* in each experiment. That is, ≈54K *unreachable* nodes were running Bitcoin at any time. Figure 5 plots the number of *responsive* nodes.

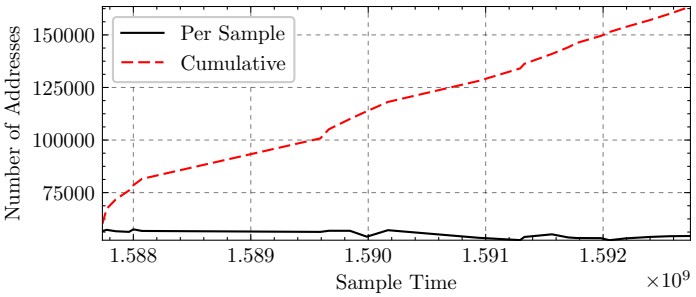


Figure 5. Unique and cumulative IP addresses of *responsive* nodes collected in each experiment. The cumulative number of *responsive* addresses follows the same trend as the *unreachable* addresses. Compared to Figure 4, the starting point on the x-axis in Figure 5 is delayed by two weeks. The difference is due to an error in the experiment, causing a delay of two weeks. On average, we collected ≈54K *responsive* addresses in each experiment.

1) Routing Attacks: In light of the new findings regarding the number of *reachable*, *unreachable*, and *responsive* nodes, we revisited the Bitcoin routing attack models [1], [22]. The most recent work on routing attacks [22] reported a high network centralization, which could be exploited using BGP attacks for network partitioning. It was shown that by hijacking 24 ASes, an adversary can isolate 50% of the Bitcoin nodes. As stated in §II, this conclusion is only valid when considering the *reachable* nodes, whereby hijacking 24 ASes results in isolating 50% *reachable* nodes.

To revisit the attack model, we conduct the same analysis as in [22], but using an updated network view. Our results show that (1) the *reachable* nodes are hosted across 2,000 unique ASes with 25 ASes hosting 50% nodes, (2) the *unreachable* nodes are hosted across 8,494 unique ASes with 36 ASes hosting 50% nodes, and (3) the *responsive* nodes are hosted across 4,453 unique ASes with 24 ASes hosting 50% nodes.

In Table I, we show the top 20 ASes that host *reachable*, *unreachable*, and *responsive* nodes. From Table I, we observe a diversity in the hosting patterns for each node type. For instance, AS4134 host 0.76% of *reachable*, 5.34% *unreachable*, and 6.18% *responsive* nodes. Therefore, for the adversary considered in [22], AS4134 will be subjectively a less preferred target, since it only hosts 218 nodes. However, if we consider the 10,104 *responsive* nodes, AS4134 becomes the second largest AS based on the number of Bitcoin nodes it hosts, and therefore a more preferred target.

This analysis clearly shows that our network view influences the steps of the routing attack, when compared to the prior work which only considered a partial view of the network. This shows that while neither of those attacks is tried on the actual Bitcoin network, an accurate characterization of the network helps determine whether the attack will work in reality.

We emphasize that for our analysis we only considered the *reachable* and *responsive* nodes, since we are confident that

Table I
TOP 20 ASES HOSTING *reachable*, *unreachable*, AND *responsive* NODES. ONLY 10 ASES ARE COMMON AMONG THE THREE CATEGORIES INDICATING THE DIVERSITY IN HOSTING PATTERN FOR EACH TYPE.

Index	ASN	% Rb	ASN	% Urb	ASN	% Resp
1	3320	8.08	3320	6.36	4134	6.18
2	24940	5.05	4134	5.34	3320	5.90
3	8881	4.60	7922	4.24	12389	4.03
4	16509	3.62	6939	3.69	4837	3.77
5	6805	2.97	8881	2.59	9009	3.28
6	14061	2.84	4837	2.28	8881	3.07
7	7922	2.55	12389	2.04	6805	2.87
8	16276	2.43	6830	1.89	3209	2.51
9	3209	2.06	3209	1.65	7922	1.56
10	12322	1.37	16509	1.54	14061	1.44
11	7545	1.33	7018	1.32	6830	1.43
12	15169	1.03	6805	1.31	3352	1.25
13	3303	0.99	9009	1.19	24940	1.18
14	6830	0.95	2856	1.14	3269	1.15
15	12389	0.94	3215	0.80	4808	1.13
16	701	0.88	4808	0.80	60068	1.12
17	20676	0.83	14061	0.78	209	1.11
18	51167	0.82	22773	0.74	7545	1.10
19	3352	0.80	1221	0.74	701	1.07
20	4134	0.76	24940	0.72	16276	0.99

they are running Bitcoin. On the other hand, the *unreachable* nodes could be (1) *reachable* or *responsive* nodes that are not running Bitcoin anymore, or (2) simply false advertisements by nodes that try to maliciously influence Bitcoin operations.

2) Impact of Unreachable Nodes: Our results show that the *unreachable* network is ≈24 times larger than the *reachable* network. Considering the prevalence of *unreachable* nodes and the size gap, we analyzed the number of *reachable* and *unreachable* addresses in all `ADDR` messages to study the impact of the *unreachable* network on the Bitcoin network.

As noted in §II, the *unreachable* addresses in the `ADDR` message provide no clear benefit to the network. The only useful information in the `ADDR` message, however, is the number of *reachable* addresses, which improves the outdegree. Therefore, propagating *unreachable* addresses may contribute to increasing the initiated (outgoing) connections failure rate.

Our results reveal that an `ADDR` message contains 14.9% *reachable* addresses and 85.1% *unreachable* addresses on average. That is, 85.1% of the addresses exchanged in the network provide no benefit. In the next section, we experimentally show how the *unreachable* IP addresses affect the network outdegree. Since the Bitcoin network topology is anonymous, we will rely on various heuristics to help our analysis.

B. Analyzing the Addressing Protocol

Our analysis showed that the *unreachable* network size is ≈24 times the *reachable* network size, and the `ADDR` messages are dominated by the *unreachable* addresses. In the following, we analyze this information along with the addressing protocol to explore the impact of those nodes on the network connectivity and synchronization.

Ideally, given that there are 10K *reachable* nodes and each node has 8 stable outgoing connections, a block could be received by all *reachable* nodes in five rounds ($8^5 > 10K$). If the number of the outgoing connections drops to 2, the block

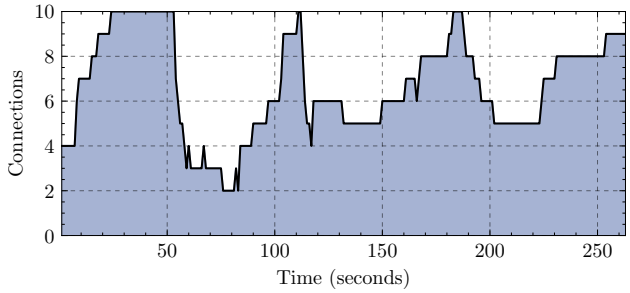


Figure 6. Results from the experiment conducted to analyze the stability of outgoing connections. The experiment was conducted for 260 seconds and we observed that the outgoing connections are highly unstable. The number of connections varied between 2–10 connections at any time.

can take up to 14 rounds ($2^{14} > 10K$) to propagate. Therefore, the stability of the outgoing connections is of our interest. To analyze the stability of the outgoing connections, and how it impacts the effective outgoing degree, and eventually impacts the time it takes to propagate a block, we deployed a Bitcoin node with the recent Bitcoin Core version (v0.20.1) and analyzed variations in the outgoing connections.

In practice, when a Bitcoin node starts, with IP addresses populated in the *new* and *tried* tables, the node selects IP addresses from both tables with equal probability and establishes outgoing connections [23], [10]. If a connection drops at any time, the process of selecting an IP address from the *new* and *tried* tables is repeated until all of the outgoing slots are complete (total of eight connections). This process naturally raises two questions. (1) *How often do outgoing connections drop?* (2) *How many outgoing connection attempts are successful?*

How often do outgoing connections drop? To answer this question, we conducted an experiment using the aforementioned Bitcoin node. Upon running the node for 260 seconds, we logged the number of the outgoing connections using the Bitcoin RPC API, and reported the results in Figure 6.

As shown in Figure 6, the numbers of the outgoing connections are highly unstable, varying between 2 and 10 connections at any time. Aside from the eight outgoing connections specified in the Bitcoin protocol, two *feeler* connections that are not used for block or transaction exchange [23] are also observed, bringing the total to 10 at times.[1] Overall, we observed that there were less than 8 connections for ≈60% of the time, while the average was 6.67 connections.

Note that the outgoing connections can be dropped for several reasons, for example, (1) the departure of a node from the network, or (2) connection/link failure in the physical network. Since those reasons are likely for all nodes in the network, we extrapolate from this observation to the network at large. We argue that the topology of the *reachable* network is constantly changing since the outgoing connections drop frequently, as shown in this experiment. This constant change can be used to reason about the high variations in network synchronization for each block Figure 1.

How many outgoing connection attempts are successful? Since now we know that the outgoing connections are unstable, the next step is to analyze the failure rate of the outgoing connection attempts. Answering this question will also help

[1]Two *feeler* connections are tried every two minutes to test if an IP address in the *new* table is *reachable*. If the connection succeeds, the IP address is moved to the *tried table* [8].

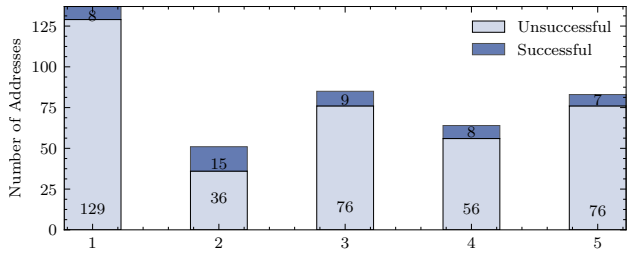


Figure 7. Results from the experiment conducted to analyze the success rate of outgoing connections. On average, only 11.2% attempts result in successful outgoing connections. For the second experiment, the number of successful connections appear to be 15. This is due the fact some connections were dropped after which the node made new successful connections.

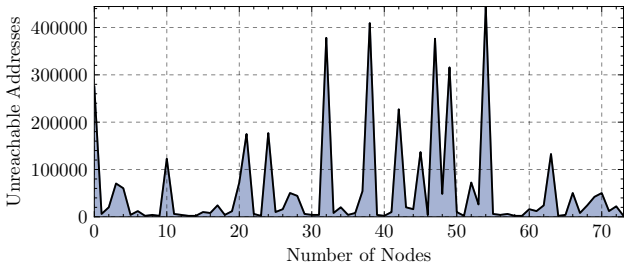


Figure 8. 73 *reachable* nodes that sent more than 1,000 addresses in the ADDR response and none of those addresses was *reachable*. Among the 73 nodes, 8 nodes sent more than *unreachable* 100,000 addresses.

us to fully understand the impact of *unreachable* addresses in the IP tables. For this purpose, we conducted five experiments in which we started the Bitcoin node and counted the total number of the outgoing connection attempts and the number of the successful connections. Each experiment was conducted for five minutes, where we restarted our node upon each experiment to maintain consistent settings.

The results of the five experiments are shown in Figure 7, where we observe a high gap between the total number of the outgoing connections and the number of successful connections. On average, only 11.2% of the connection attempts were successful. Moreover, in one experiment, only 8 out of 137 attempts (5.8%) were successful. Furthermore, Figure 7 shows a high diversity in the total number of the outgoing connections in each experiment. In two experiments, the total number of the outgoing slots was not filled during the experiment duration. In the second experiment (Figure 7), the total number of successful connections was 15. Since the maximum outbound connections is only 8 (except the *feeler* connections), this shows that some successful connections dropped and the node tried new connections from the IP tables.

Our results show a high outgoing connection failure rate. The failure rate of 88.8% comes as a surprise, shedding light on weaknesses in the addressing protocol. Ideally, each outgoing connection should be successful. However, the low success rate in the outgoing connections shows that the network condition is far from ideal. This is in part due to the *unreachable* addresses dominating the nodes' IP tables. Moreover, since the outgoing connections are unstable, our results also show that even the *reachable* nodes may leave the network. Therefore, the outgoing connections to the *reachable* nodes that have left the network also contribute to the low success rate. From Figure 6 and Figure 7, we conclude that the *unreachable* network adversely affects the *reachable* network topology since the average node outdegree is less than the default outdegree. Extrapolating this behavior to other nodes in the network means that the average network outdegree is below the expected outdegree, and the block propagation in the network takes longer than expected, thus affecting network synchronization. Our experiments also reveal that this undesirable network state is caused by the weakness in the Bitcoin addressing protocol that allows the propagation of *unreachable* IP addresses in the ADDR message.

Malicious Peer Behavior. Since the weaknesses in the addressing protocol is clear, it is also possible that malicious nodes are exploiting them to weaken the network connectivity. For example, a malicious node can propagate only *unreachable* addresses in the ADDR message to flood the IP tables of the receiving nodes. In this section, we report how we detected the *reachable* nodes that were carrying out such activities.

To set out, we first outline a heuristic for detecting the *reachable* malicious nodes. Note that in each ADDR message, there must be at least one *reachable* address because the sending node always sends its own address in the ADDR message. Moreover, the *reachable* node must be connected to at least 1 to 8 other *reachable* nodes to fill their outgoing connection slots. As such, if there is no *reachable* address among all IP addresses in the ADDR response, we can consider the sending node to be malicious.

From our dataset, we identified 73 *reachable* nodes that only sent *unreachable* IP addresses in all ADDR messages. Among them, 8 nodes sent more than 100,000 *unreachable* addresses in the ADDR messages. One node sent more than 400,000 IP address in the ADDR message, none of which was a *reachable* address. In Figure 8, we plot the number of *unreachable* IP addresses sent by those malicious nodes. From a network standpoint, these 73 nodes are harmful, since they flood the IP tables of their connections with *unreachable* addresses. As discussed earlier, if IP tables are dominated by the *unreachable* addresses, the outgoing connection success rate decreases significantly. A closer inspection reveals that 43 nodes (59%) are hosted in the same AS (AS3320).

Key Takeaways. From our analysis so far, we have the following observations. First, the current Bitcoin addressing protocol can be overwhelmed by the size of *unreachable* network, which results in unstable network conditions and affects block propagation. Therefore, the *unreachable* network size and the Bitcoin addressing protocol, considered simultaneously, can significantly influence the network synchronization. Second, there are various malicious activities in the *reachable* network that exploit weaknesses in the addressing protocol. In §V, we leverage these insights to propose refinements to the Bitcoin Core design.

C. Information Relaying Protocol

As discussed in §I, the theoretical models of Bitcoin [15], [7], [20] assume that a Bitcoin node concurrently releases every new block to all of its connections. As such, if a Bitcoin node has all 125 connection slots filled and the node produces a block, the block should be relayed to all 125 connections simultaneously. In contrast, if there is a delay in relaying the block to each connection, the nodes receiving the block earlier will synchronize faster. In this section, we analyze the practical implications of the block relaying protocol in Bitcoin and how those aspects affect the network synchronization.

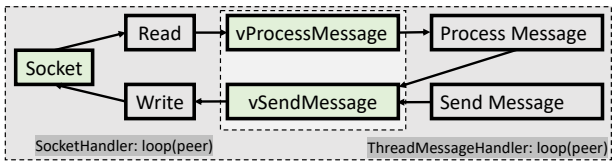


Figure 9. Message handling workflow. The `SocketHandler` thread loops over each peer and reads incoming messages into the `vProcessMsg` queue. It also sends outgoing messages from the `vSendMessage` queue. The `ThreadMessageHandler` thread reads messages from the `vProcessMsg` queue and sends the output to the `vSendMessage` queue.

Algorithm 3: Processing P2P Messages

Input: $\mathbb{C}_{y,z}$ where y and z are the number of connections and messages sent by each connection
foreach $y \in \mathbb{C}_y$ **do**
  foreach $z \in \mathbb{C}_{y,z}$ **do**
    if no new block **then**
      invoke `ThreadMessageHandler()` and `ThreadMessageHandler()`
      process message $\mathbb{C}_{y,z}$ (see Figure 9)
    if new block **then**
      append block to `vSendMessage` queue
      increment z
      process message $\mathbb{C}_{y,z}$ (see Figure 9)
return empty queue $\mathbb{C}_{y,0}$

For this purpose, we inspect the *net.cpp* file in the Bitcoin Core source code [3]. We found that when the Bitcoin Core starts, it creates two threads to handle the protocol messages. On the one hand, the `SocketHandler` thread reads the incoming messages from a connected peer and stores them in the `vProcessMsg` queue. It then sends the outgoing messages to the peer from the `vSendMessage` queue using a `Write` buffer. The `ThreadMessageHandler` thread reads the messages from `vProcessMsg` and processes them using the `ProcessMessage` function.

We use the reconstructed protocol information from our analysis to sketch the Bitcoin Core message handling procedure in Figure 9. We further illustrate this workflow with an example. Assume two connected Bitcoin nodes, A and B, where A sends a `GETADDR` request to B. At B: (1) the `SocketHandler` thread will queue the `GETADDR` message in the `vProcessMsg` queue, (2) the `ThreadMessageHandler` thread will read the message from the`vProcessMsg` and generate the `ADDR` response using the `ProcessMessage` function, (3) the `ProcessMessage` function will send the `ADDR` response to `vSendMessage`, and (4) the `SocketHandler` thread will write the response to the socket connected with A. However, if during this process, B also generates a new block and wants to send it to A, the `SendMessages` function will queue the block behind the `ADDR` message in `vSendMessage`, and the `ThreadMessageHandler` thread will send it to A after sending the `ADDR` message.

Our analysis also revealed that a Bitcoin node schedules the connections in a round-robin manner, processing one message per socket for each connection. For instance, if node B is connected to five other nodes (A–F) and wants to send a block, B will loop over each connection to send that block.

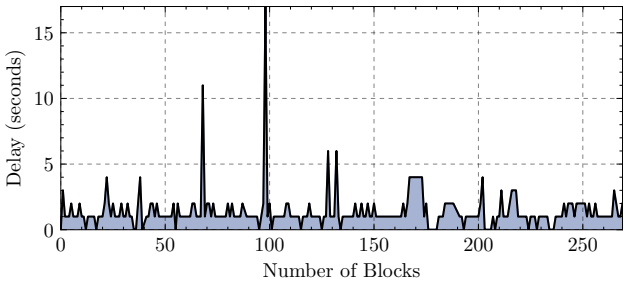


Figure 10. Delay between the time of receiving the block and the time at which the block is relayed to the last connection. On average, it takes 1.39 seconds to relay blocks to all connections

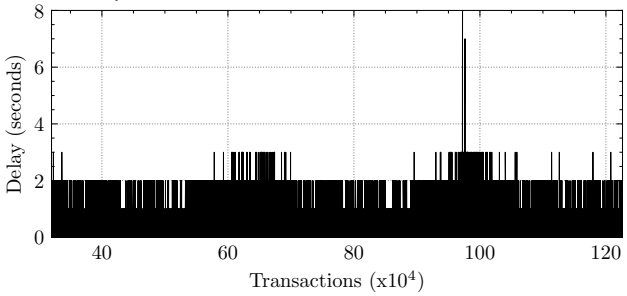


Figure 11. Delay between the time of receiving the transaction and the time at which the transaction is relayed to the last connection. On average, it takes 0.45 seconds to relay transaction to all connections

Therefore, the block relaying in Bitcoin does not follow a broadcast model assumed in the prior works.

Moreover, if B wants to send a block to A, and A has already sent 3 `GETADDR` requests, then B will process one request per loop per connection. As a result, A will get the block after B processes 15 requests for all connections including A's three `GETADDR` requests. In algorithm 3, we show the pseudo-code processing the information relaying in Bitcoin Core.

Empirical Evaluation. A natural effect of the round-robin relaying is that some connections receive blocks earlier than others. Therefore, there is a time gap between when a node receives a block from the network, and when the node relays that block to all its connections. In Bitcoin, round-robin relaying is not limited to blocks only, but also applies to transactions. We note that transaction relaying also plays a critical role in the network synchronization, due to the deployment of the "compact block" relay method in Bitcoin [4].

In the compact block relay method, a node only receives the block header and transaction identifiers from the sending node. The node then reconstructs the block using transactions from its memory pool [4]. If some transactions are missing from the memory pool, the node requests those transactions from the sending node to fully reconstruct the block. If those transactions are delayed, the node cannot reconstruct the block and remains behind the blockchain. Therefore, transaction relaying also plays a significant role in the network synchronization. Taking this into account, we measure the delay introduced by the round-robin transaction and block relaying.

To empirically measure the block relaying delay, we set up a *reachable* Bitcoin node with 8 outgoing connections and 17 incoming connections. We then measure (1) the time when the node receives a transaction or a block from the network, and (2) the time when that transaction or block is relayed to the last connection. We call the difference between the two events as the relaying time. Naturally, a high relaying time is undesirable for network synchronization since the receiving node stays

behind the blockchain during that period. We collected the relaying time for transactions and blocks from the *debug.log* file in the Bitcoin Core data directory. The log file captures each event at one second interval. If a transaction or a block is received and relayed within a second, the timestamp for the two events is the same.

Figure 10 and Figure 11 report the results collected by our node over two days. Figure 10 shows that the average relaying time for all blocks was 1.39 seconds, with a minimum and a maximum relaying time of 0 and 17 seconds, respectively. Figure 11 shows that the average relaying time for all transactions was 0.45 seconds, with a minimum and a maximum relaying time of 0 and 8 seconds, respectively. From these results, we conclude that the round-robin relaying adds delay in relaying transactions and blocks to the nodes, thus affecting network synchronization. We further observed that during the relaying process, the sending node does not prioritize the *reachable* nodes over the *unreachable* nodes; a distinction that can be easily made by observing the incoming and outgoing connections. As mentioned in §I, *reachable* nodes enable network synchronization by relaying blocks to other *reachable* nodes. As such, if the *reachable* nodes are among the last connections to receive transactions or blocks, the network synchronization is affected considerably; e.g., delay of up to 17 seconds in some cases, as shown in Figure 10.

D. Network Churn

Due to the *permissionless* nature of Bitcoin, nodes can leave the network at any time, causing the network churn. If *reachable* nodes leave the network, the average network outdegree decreases, which affects network synchronization. Hereafter, we measure the churn impact on network synchronization.

If a node leaves the network, at least 8 outgoing connections drop in the *reachable* network [11], [14]. Although, intuitively, it might appear that if the number of *reachable* nodes in the network is constant, the average network outdegree must be constant as well. However, this assumption can be false once we take the churn into account.

To illustrate this problem, consider a *reachable* node A with 8 stable outgoing connections. Next, assume that the node A leaves the network at time t_x, and all its outgoing connections drop. Further assume that at the same time t_x, another *reachable* node B joins the network. As shown in Figure 7, a node takes time to successfully establish 8 outgoing connections. If we assume that node B successfully makes 8 stable outgoing connections by the time t_y, then during the time gap $\Delta = t_y - t_x$, the network will have a fewer number of outgoing connections despite the same network size. Accordingly, the network will have a smaller outdegree during $\Delta = t_y - t_x$.

Moreover, even after establishing 8 outgoing connections, node B will take time to catch up with the blockchain by requesting blocks that are not present in its local blockchain. Once node B synchronizes with an up-to-date blockchain, only then it can help relay the latest blocks to other *reachable* nodes. In other words, during the time when node B is not up-to-date, it does not contribute to the network synchronization. Moreover, the process of catching up with the blockchain can take a few days if node B joins the network for the first time.

Algorithm 4: Creating Binary Matrix for Churn

```
Input: Reachable addresses U and sampling time T
Initialize: Binary matrix M
foreach ip i ∈ N do
    foreach sample j ∈ T do
        if ip in sample then
            M_{i,j} ← 1
        if ip not in sample then
            M_{i,j} ← 0
return Binary matrix M
```

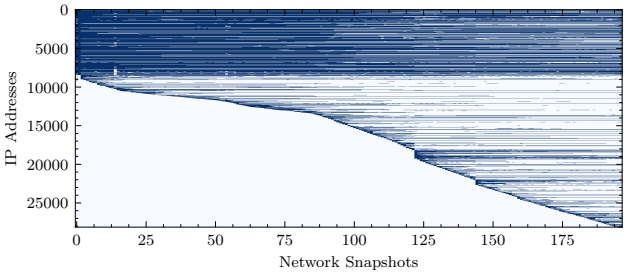


Figure 12. Plot of the binary matrix in algorithm 4. Value 1 is marked 1 while value 0 is marked white. For a given IP address, end-to-end horizontal line shows that the address was connected in each experiment.

To evaluate a node's capability for contributing to the network synchronization, we set up a Bitcoin node with an up-to-date blockchain. In the *debug.log* file, we observed that our node was relaying the latest blocks to its connections, thus helping other nodes to synchronize. Next, we restarted the node and recorded the time the node took to synchronize again and start to relay blocks to its connections. Our results show that the node took 11 minutes and 14 seconds to synchronize with the network and regain the ability of relaying blocks to its connections. Most of this time was spent on establishing stable outgoing connections and synchronizing on the latest block. Alternatively, if the node stayed offline for several days, it would have taken a longer time to synchronize.

From our results, we concluded that the departure of synchronized nodes and arrival of new nodes are unfavorable for network synchronization. With the departure of existing nodes and the arrival of new nodes, a high churn rate leads to poor synchronization. In the following, we model the Bitcoin churn and measure the nodes arrival and departure rates.

Modeling Network Churn. To model churn in the *reachable* network, we systematically evaluate (1) the arrival time of a new *reachable* node, (2) the departure time of that node, and (3) whether a node rejoins the network after departure.

For this analysis, we sampled all of the *reachable* addresses (N_r in algorithm 1) as an object T, with the network sampling time as the object keys and the *reachable* IP addresses corresponding to the sampling time as values. Next, we collected all 28,781 unique *reachable* IP addresses in a list U, and checked the presence of each IP address for the sorted keys in T. We then initialized a zero matrix $\mathbf{M}$ in which each row denoted an IP address in U, and each column denoted the sorted sampling time from T. If an address was found at the sampling time, its index in $\mathbf{M}$ is changed from 0 to 1. In algorithm 4, we outline the procedure of obtaining the binary matrix $\mathbf{M}$, and in Figure 12, we plot its binary image to provide a high level overview of churn in the *reachable* network. The colored

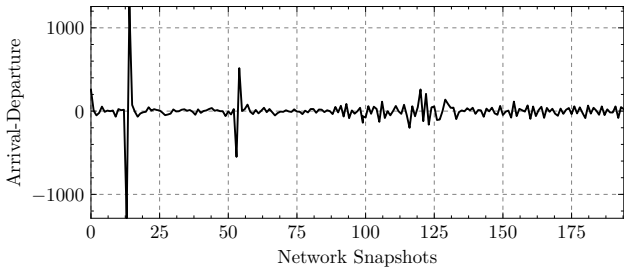


Figure 13. The difference between the number of nodes that leave the network and the new nodes that join the network.

region shows the presence of an IP address in the network.

From Figure 12, we observe the following. (1) A dominant white region in the bottom left shows that a significant number of new nodes join the network. (2) A majority of lines does not cover the entire x-axis after the starting point, indicating that a significant number of nodes leave the network. (3) The reappearance of a few lines on the x-axis shows that some nodes rejoin the network after leaving. (4) A few lines covering the entire x-axis show that a few nodes are always present in the network. More precisely, we found 3,034 nodes that did not leave the network throughout our measurement.

We empirically analyze nodes that join and leave the network daily. For that purpose, we take two consecutive network snapshots and count the addresses that change in them by comparing each column in $\mathbf{M}$ (algorithm 4) with the previous column. A change in the row value from 1 to 0 indicates a node departure, while a change from 0 to 1 indicates a node arrival. In Figure 13, we report our results showing that the difference between the arrival of new nodes and departure of existing ones is small. We also note that ≈708 nodes (8.6% *reachable* nodes) leave the network everyday, replaced by an equal number of new nodes. As explained earlier, replacing departing nodes with new nodes affects synchronization, since the new nodes take time to receive the blockchain before contributing to synchronization. Therefore, the arrival rate of 8.6% shows that a significant number of non-synchronized nodes appear in the Bitcoin network each day.

Measuring Departure of Synchronized Nodes. Among the *reachable* nodes that leave the network, not all nodes are synchronized with an up-to-date blockchain. Moreover, if a non-synchronized node leaves the network, the average network synchronization would increase. Therefore, a logical question would be to determine *how many synchronized nodes leave the network?* By contrasting the departure of synchronized nodes between 2019 and 2020, we can confidently answer the problem of decreasing synchronization shown in Figure 1.

In order to experimentally evaluate the departure rate of the synchronized nodes, we used the Bitnodes dataset which we have been collecting since September 2019, at 10 minutes interval. To contrast our results with Figure 1, we divided our dataset into two segments, consistent with the segmentation used for Figure 1 . The first segment included data from September to December 2019, and the second segment included data from January to April 2020. For each segment, among the total number nodes that leave the network in 10 minutes, we counted the number of synchronized nodes.

Our results show that the average number of synchronized nodes that left the network in 10 minutes was 3.9 (≈4) in 2019, which then increased to 7.6 (≈8) in 2020. In other words, the departure of synchronized nodes nearly doubled in 2020, thus decreasing the total number of nodes that contribute to network synchronization. Since the Bitcoin addressing protocol and the block relaying protocol did not change between 2019 and 2020, it is safe to assume that their impact on network synchronization has remained constant. As a result, we find that the most significant change in the network is the churn among the synchronized nodes, which has doubled in 2020, leading to a deteriorating network synchronization.

Key Takeaways. Our analysis shows increasing churn among the synchronized nodes, especially in 2020. When the synchronized nodes leave the network, the outgoing connections of their peers drop below the default threshold, upon which those nodes try new connections from their IP tables. Due to the weaknesses in the addressing protocol, the IP tables of *reachable* nodes are dominated by *unreachable* addresses, causing a high failure rate of the outgoing connections (Figure 7). Due to the departure of synchronized nodes and delayed connection recovery of their peers, fewer nodes are left in the network that contribute to network synchronization. Moreover, during the failed connection attempts, the average network outdegree remains low, further slowing down the block propagation.

We also observed 73 malicious nodes propagating *unreachable* addresses in the network to overwhelm the IP tables of other *reachable* nodes and increase the connection failure rate. Moreover, the information relaying protocol does not prioritize block propagation to the *reachable* nodes over *unreachable* nodes, resulting in block relaying delay by up to 17 seconds in some cases (Figure 10).

V. Improving Bitcoin Network Synchronization

Based on our measurements and analysis, in this section, we propose refinements to the Bitcoin Core design in order to improve the network synchronization.

Refining the Addressing Protocol. Since the Bitcoin network is *permissionless*, we cannot prevent the invariant network churn, despite its implications on network synchronization. However, we can help the network to recover from the departure of synchronized nodes by tailoring the addressing protocol. Since relaying *unreachable* addresses in the `ADDR` message does not suit the network, the source code can be modified to only select IP addresses from the `tried` table for the `ADDR` message. This will significantly improve the success rate of outgoing connections.

Refining the `tried` Table. We note that selecting IP addresses from the `tried` table only partially solves the problem. The churn analysis shows that 708 *reachable* nodes leave the network everyday. Despite their departure, their IP addresses stay in the `tried` of their connections. A *reachable* IP address is removed from the `tried` table if: (1) more than 10 connection attempts to that address fail in one week, or (2) the IP address is in the table for 30 days [10]. As such, even if we apply the policy of sending IP addresses from the `tried` table, if the sending node has not evicted the IP address from `tried` table, it will not be useful for the receiving node.

To make the IP address eviction policy more efficient, we revisit the threshold of retaining an IP address in the `tried` table for 30 days. In the Bitcoin Core source code, we did not find any justification for retaining an IP address for 30 days. We believe the underlying assumption of the early

developers was that the average network lifetime for a majority of nodes will be 30 days. To revise this policy, we use results from Figure 12 to provide a more realistic estimate of a node's lifetime. Our results show that the average network lifetime of a node is only 16.6 days. Therefore, by reducing the limit of 30 days to 17 days, we can increase the eviction rate of IP addresses that leave the network. With an increased eviction rate, we can increase the percentage of *reachable* addresses in the `ADDR` message, thus increasing the success rate of outgoing connections.

Prioritizing Block Relay. Network synchronization can also be improved by prioritizing block relaying to the *reachable* nodes. In the current implementation, a sending node does not distinguish between incoming connections and the outgoing connections. While incoming connections can be from both *reachable* and *unreachable* nodes, the outgoing connections are always established with the *reachable* nodes. As such, if a new block is mined or received from any connection, the node should first relay that block to all the outgoing connections, thereby increasing block propagation among the *reachable* nodes. Moreover, if there is a queue of requests (*i.e.,*`GETADDR` messages) in the `vSendMessage` queue, then the block can be prioritized over those requests to minimize unusual delay in block relaying (*i.e.,* 17 seconds in Figure 10).

By incorporating these changes, the Bitcoin network synchronization can be significantly improved despite the increasing network churn. If a synchronized node leaves the network, all its peers will immediately recover their outgoing connections, thus maintaining the average network outdegree. If the synchronized node does not rejoin the network, its IP addresses will be removed from the `tried` tables in 17 days. This will prevent the undesirable relaying of the address in the `ADDR` message. Finally, by prioritizing block relaying to the outgoing connections, we can ensure that the *reachable* network synchronizes quickly over a newly published block. By maintaining the network outdegree and ensuring faster block relay, the undesirable impact of churn on network synchronization can be mitigated.

VI. Conclusion

In this paper, we have conducted the root cause analysis of the deteriorating Bitcoin network synchronization. Through measurements and analysis, we found that the deteriorating network synchronization is due to (1) a large and increasing number of *unreachable* nodes, (2) weaknesses in the network addressing protocol, (3) the delay introduced by round-robin block relaying, and (4) the high churn among the *reachable* nodes. Among them, the most significant factor in the recent months is the churn among the synchronized *reachable* nodes. Finally, based on our measurements and analysis, we have proposed refinements to the current Bitcoin Core implementation in order to improve the network synchronization.

VII. Acknowledgement

M. Saad and D. Mohaisen were supported in part by NRF grant NRF-2016K1A1A2912757. S. Chen was supported in part by NSF under grant CNS-2007153.

References

[1] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *IEEE Symposium on Security and Privacy*, 2017, pp. 375–392, https://doi.org/10.1109/SP.2017.29.

[2] B.Community, "Bitnodes: Discovering all reachable nodes in bitcoin." [Online]. Available: https://bitnodes.earn.com/

[3] B. Community, "Bitcoin core version history," 2018, https://bitcoin.org/en/version-history.

[4] M. Corallo, "Bitcoin improvement proposal 152." [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[5] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE International Conference on Peer-to-Peer Computing*, 2013, pp. 1–10, https://doi.org/10.1109/P2P.2013.6688704.

[6] 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 *International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, vol. 11598. Springer, 2019, pp. 550–566. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7_32

[7] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol with chains of variable difficulty," in *International Cryptology Conference on Advances in Cryptology*, 2017, pp. 291–323. [Online]. Available: https://doi.org/10.1007/978-3-319-63688-7_10

[8] E. Heilman, "Feeler connections," https://github.com/bitcoin/bitcoin/pull/9037, 2018.

[9] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *ISOC Network and Distributed System Security Symposium*, 2017, https://bit.ly/3s5mjH1.

[10] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security Symposium*, 2015, pp. 129–144. [Online]. Available: https://bit.ly/3bdJCZ0

[11] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the bitcoin network: Characterization and impact," in *IEEE International Conference on Blockchain and Cryptocurrency*, 2019, pp. 431–439. [Online]. Available: https://doi.org/10.1109/BLOC.2019.8751297

[12] L. D. Jr, "Seeds.txt file," https://bit.ly/3d7QOXc, Feb 2020, (Accessed on 08/31/2020).

[13] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: bitcoin lightweight client privacy using trusted execution," in *USENIX Security Symposium*, 2019, pp. 783–800. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/matetic

[14] S. G. Motlagh, J. V. Misic, and V. B. Misic, "Modeling of churn process in bitcoin network," in *IEEE International Conference on Computing, Networking and Communications*, 2020, pp. 686–691. [Online]. Available: https://doi.org/10.1109/ICNC47757.2020.9049704

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.

[16] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 817–831. [Online]. Available: https://doi.org/10.1145/3319535.3354237

[17] T. Neudecker, "Characterization of the bitcoin peer-to-peer network (2015-2018)," http://dsn.tm.kit.edu/bitcoin/publications/bitcoin_network_characterization.pdf, 2019.

[18] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing*, 2016, pp. 358–367. [Online]. Available: https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070

[19] S. Park, S. Im, Y. Seol, and J. Paek, "Nodes in the bitcoin network: Comparative measurement study and survey," *IEEE Access*, vol. 7, pp. 57 009–57 022, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2914098

[20] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," *IACR Cryptology ePrint Archive*, vol. 2016, p. 454, 2016. [Online]. Available: http://eprint.iacr.org/2016/454

[21] L. Ren, "Analysis of nakamoto consensus," Cryptology ePrint Archive, Report 2019/943, 2019, https://eprint.iacr.org/2019/943.

[22] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on bitcoin: Colliding space, time, and logic," in *IEEE International Conference on Distributed Computing Systems*, 2019, pp. 1175–1187. [Online]. Available: https://doi.org/10.1109/ICDCS.2019.00119

[23] 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 *IEEE Symposium on Security and Privacy*, 2020, pp. 894–909. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00027