SyncAttack: Double-spending in Bitcoin Without Mining Power

Muhammad Saad
University of Central Florida
Orlando, USA
saad.ucf@knights.ucf.edu

Songqing Chen
George Mason University
Virginia, USA
sqchen@gmu.edu

David Mohaisen
University of Central Florida
Orlando, USA
mohaisen@ucf.edu

ABSTRACT

The existing Bitcoin security research has mainly followed the security models in [22, 35], which stipulate that an adversary controls some mining power in order to violate the blockchain *consistency* property (*i.e.,* through a double-spend attack). These models, however, largely overlooked the impact of the realistic network synchronization, which can be manipulated given the *permissionless* nature of the network. In this paper, we revisit the security of Bitcoin blockchain by incorporating the network synchronization into the security model and evaluating that in practice. Towards this goal, we propose the ideal functionality for the Bitcoin network synchronization and specify bounds on the network outdegree and the block propagation delay in order to preserve the *consistency* property. By contrasting the ideal functionality against measurements, we find deteriorating network synchronization reported by Bitnodes and a notable churn rate with ≈10% of the nodes arriving and departing from the network daily.

Motivated by these findings, we propose SyncAttack, an attack that allows an adversary to violate the Bitcoin blockchain *consistency* property and double-spend *without using any mining power*. Moreover, during our measurements, we discover weaknesses in Bitcoin that can be exploited to reduce the cost of SyncAttack, deanonymize Bitcoin transactions, and reduce the *effective* network hash rate. We also observe events that suggest malicious nodes are exploiting those weaknesses in the network. Finally, we patch those weaknesses to mitigate SyncAttack and associated risks.

CCS CONCEPTS

• **Security and Privacy** → Distributed systems security.

KEYWORDS

Distributed Systems, Nakamoto Consensus, Security and Privacy

ACM Reference Format:
Muhammad Saad, Songqing Chen, and David Mohaisen. 2021. SyncAttack: Double-spending in Bitcoin Without Mining Power. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3460120.3484568

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484568

1 INTRODUCTION

The double-spend attack is a classical example of violating the Bitcoin blockchain *consistency* [22, 29, 34], whereby an adversary who controls some mining power forks the public chain with a longer private chain [24, 31]. The double-spend attack succeeds with a high probability if the adversary controls 51% of the network's hash rate [19, 27]. Since acquiring 51% hash rate in Bitcoin is prohibitively costly [3], various attack optimizations have been proposed to reduce this requirement by exploiting the physical network characteristics, including the biased distribution of the mining nodes [6, 38] and the block propagation delay [19, 34].

Despite leveraging the physical network characteristics to the adversary's advantage, the optimized double-spend attacks [19] still use the Nakamoto's attack construction [31] as a blue print to model a block race between the adversary and the honest miners. Therefore, their threat models inherit two specifications from [31]: (1) the adversary controls at least some mining power, and (2) there is a stable hash rate distribution during the attack (*i.e.,* miners do not join or leave the network [24]). The probability of a successful double-spend attack varies exponentially if the hash rate distribution changes during the attack [36].

In contrast, in this paper we find that an unstable hash rate distribution can be used instead to the adversary's advantage, irrespective of new miners joining the network. Moreover, variations in the hash rate are exploited to exempt the adversary from using the mining power altogether, while still double-spending successfully.

Our findings are based on the two characteristics of the real-world Bitcoin network that have not been thoroughly explored in the past. First, we note that the blockchain forks, antecedent to a double-spend attack, do not solely rely on the hash rate distribution among the honest miners. In fact, forks can also occur due to weak network synchronization that characterizes the blockchain view of each node in the network [35]. We further observe that the network synchronization depends on the overlay topology of *reachable* nodes (e.g., using public IP addresses; see §2 for details) in the network and the block propagation delay among those nodes. If the overlay topology partitions or the block propagation delay increases, the blockchain can fork even in the absence of an adversary [35]. Therefore, the network synchronization plays a key role in preserving the blockchain *consistency*. However, despite its significance, the network synchronization has not been comprehensively characterized in the Bitcoin security model.

Second, we note that prior security models [22, 35] did not fully incorporate the *permissionless* nature of Bitcoin, which is intrinsic in its design. The *permissionless* network allows nodes to leave or join the network at any time, thereby causing the network churn. Our study reveals that the network churn can be exploited to deteriorate

the network synchronization and launch new forms of partitioning attacks to disrupt the overlay topology. [1]

Although partitioning attacks on the Bitcoin network have been extensively studied [5, 6, 38, 42], they have not been examined in the context of its *permissionless* nature and the associated churn. In particular, the partitioning attacks that involve manipulating the incoming and outgoing connections of a Bitcoin node assume (1) the node persistently stays in the network for many days [42], and (2) the adversary controls more than 100K IP addresses to poison the IP tables of that node [26, 42]. In §4.2, we show that the ongoing network churn might affect the first assumption, and in §4.2.1, we show that the second assumption can be costly.

Although the network churn makes the existing partitioning attacks infeasible, it also provides new partitioning opportunities to split the network. We show that by setting up 10 *reachable* nodes and 102 *unreachable* nodes, an adversary can launch SyncAttack to split the network between the existing nodes and the newly arriving nodes. By controlling the communication between the two partitions, the adversary can deteriorate the network synchronization and double-spend, without using any mining power.

Contributions and Roadmap. In short, our work examines the network synchronization in the Bitcoin security model and analyzes its robustness in the *permissionless* settings. Additionally, by measuring and characterizing the behavior of the real world Bitcoin network, we propose SyncAttack, an attack that allows an adversary to deteriorate the network synchronization and launch double-spend attacks without mining power. The highlights of our contributions are as follows.

(1) We present the first ideal functionality for the Bitcoin network synchronization that characterizes the blockchain security in light of the overlay network topology and the block propagation delay (§3).
(2) We conduct measurements to analyze the Bitcoin network synchronization in the real world (§4). Our measurements reveal a deteriorating network synchronization resulting in an increasing number of forks.
(3) We characterize the *permissionless* nature of the Bitcoin network by measuring the churn among the *reachable* nodes (§4.2). Our measurements show a notable network churn rate where 10% of the *reachable* nodes leave the network every day, replaced by an almost equal number of new nodes.
(4) We incorporate the churn in the Bitcoin security model and show that if mining nodes experience the churn, then an adversary can launch SyncAttack to partition them and double-spend without using any mining power (§5).
(5) A byproduct of our study is the discovery of Bitcoin weaknesses that can be exploited to (1) deanonymize transactions, and (2) reduce the *effective* hash rate of the mining nodes. Our experiments on the testnet reveal that Bitcoin transactions can be deanonymized with ≈79.4% accuracy. In §6, we propose SyncAttack countermeasures and strengthen Bitcoin Core to resist transaction deanonymization.[2]

Additionally, the paper includes background in §2, related work in §7, conclusion in §8, and appendices in §A–§E.

2 BACKGROUND

In this section, we succinctly review the core concepts related to the Bitcoin network synchronization. We start by outlining the Bitcoin network anatomy and different node types.

Network Structure. The Bitcoin network consists of nodes connected in a Peer-to-Peer (P2P) model. Those nodes exchange transactions, order transactions in a block, and execute the Nakamoto consensus to mine a *valid* block. Once a node mines a block, the block is relayed to the rest of the network.

Full Nodes and SPV Nodes. Broadly speaking, there are two types of nodes in the Bitcoin network, the full nodes and the "Simplified Payment Verification" (SPV) nodes [30]. The full nodes maintain a complete Bitcoin blockchain ledger while the SPV nodes only maintain the block headers and request Merkle proofs from full nodes to verify transactions.

***Reachable* and *Unreachable* Full Nodes.** Full nodes can be further categorized into *reachable* and *unreachable* nodes. The *reachable* nodes establish outgoing connections to other *reachable* nodes, and accept incoming connections from both *reachable* and *unreachable* nodes. On the other hand, *unreachable* nodes (often behind NATs [20]) only establish outgoing connections to the *reachable* nodes and do not accept incoming connections. By default, *reachable* nodes establish 10 outgoing connections and accept up to 115 incoming connections, while *unreachable* nodes only establish 10 outgoing connections. Since *unreachable* nodes drop incoming connections, no two *unreachable* nodes can directly connect to each other. Information exchange (*i.e.,* blocks) between two *unreachable* nodes is enabled through the *reachable* nodes.

Mining and Non-mining Nodes. In the last few years, the mining difficulty in Bitcoin has significantly increased, limiting the mining capability to a few mining pools that use specialized hardware (*i.e.,* ASIC mining rigs [41]) to mine blocks. After mining a block, the mining pools use either *reachable* or *unreachable* nodes to relay it in the P2P network. The nodes used by the mining pools for block relaying are also called the mining nodes [37] or the mining pool gateways [20]. Prior studies show that at any time, there are ≈6K-10K [6, 38] *reachable* nodes in the Bitcoin network, with ≈100–350 among which being the mining nodes [20, 37].

As noted in [20, 37], another key difference between the mining and non-mining nodes is that the mining nodes have a long network lifetime and do not experience significant churn. However, in 2021, changes in the Bitcoin price have also caused behavior changes in the mining pools. For instance, new mining pools including ArkPool, SBI Crypto, and Foundry USA joined the network between 2020 and 2021, while others including PHash.io, NovaBlock, and BytePool left the network during that time period [17][3]. Given the increase in the number of mining pools joining and leaving the Bitcoin network, it is fair to assume that compared to the previous years, in 2021, churn among the mining nodes has increased [17, 20, 37].

[1] Our characterization of the *permissionless* network acknowledges the possibility of a change in the mining power distribution due to the arrival or departure of miners.

[2] **Responsible Disclosure.** We have notified the community about the discovered weaknesses and shared our patch code. The experiments related to SyncAttack were only conducted on our own Bitcoin nodes. We did not manipulate connections of other *reachable* nodes in the Bitcoin network.

[3] A complete list of all the notable mining pools that joined or left the network in the last few months can be found in [17].

Ideal Functionality $\mathcal{F}_{\mathsf{syn}}$

Input: *Reachable* nodes N_r, with each $n_i \in N_r$ establishing O_i outgoing connections and accepting I_i incoming connections. The average network outdegree $\deg^+(N_r)$ is greater than the minimum network outdegree $\deg^+_{\min}(N_r)$ (see §A) to form a connected overlay topology [40]. The mining power H is uniformly distributed among N_r such that $\sum_{\forall i} h_i = 1$, where h_i is the mining power of n_i. Each $n_i \in N_r$ maintains a blockchain ledger C, and participates in the block mining race which proceeds for l rounds. The mining race is arbitrated by a trusted party $\mathcal{F}_{\mathsf{syn}}$ which knows N_r, H, $\deg^+(N_r)$, and $\deg^+_{\min}(N_r)$. In each round, the trusted party $\mathcal{F}_{\mathsf{syn}}$ observes the following states.

Start: Each $n_i \in N_r$ starts mining on C with b_r as the latest block. The probability of mining the next block b_{r+1} is h_i/H. If n_i successfully mines $b_{r+1} \preceq b_r$ ($\preceq$ is the prefix relationship [22, 34]), n_i appends b_{r+1} to C and relays b_{r+1} to O_i, I_i, $\mathcal{F}_{\mathsf{syn}}$, and moves to the next round.

Receive: Consistent with the current Bitcoin protocol [31], if a node n_i receives two valid blocks $b_{r+1} \preceq b_r$ and $b'_{r+1} \preceq b_r$ in any round, n_i will stop its computation and start mining on the block that it receives the earliest. For instance, if $b_{r+1} \preceq b_r$ is received at t_1 and $b'_{r+1} \preceq b_r$ is received at t_2, where $(t_1 < t_2)$, then n_i mines on $b_{r+1} \preceq b_r$. Additionally, n_i forms two chains $C_1 \leftarrow b_{r+1} \preceq b_r$ and $C_2 \leftarrow b'_{r+1} \preceq b_r$, with C_1 as the dominant chain on which n_i mines. Then, n_i, relays $b_{r+1} \preceq b_r$ to O_i and I_i, and moves to the next round.

Propagate: A valid block $b'_{r+1} \preceq b_r$ takes $k = \log_{\deg^+(N_r)}|N_r|$ steps to reach all the *reachable* nodes. Each step adds a fixed delay t, such that kt is the end-to-end delay for $b'_{r+1} \preceq b_r$ to end up in C of each n_i. We enforce the end-to-end delay kt within a bound by threshold parameter T such that $kt \leq T$ to prevent forks during block propagation.

Evaluate: Once $\mathcal{F}_{\mathsf{syn}}$ receives a valid block $b_{r+1} \preceq b_r$, it checks if the network satisfies the synchronization property. For that purpose, $\mathcal{F}_{\mathsf{syn}}$ first checks if the network outdegree is greater than the minimum outdegree ($\deg^+(N_r) > \deg^+_{\min}(N_r)$). $\mathcal{F}_{\mathsf{syn}}$ then concludes that $b_{r+1} \preceq b_r$ will eventually reach all $n_i \in N_r$ if the condition is satisfied. Next, $\mathcal{F}_{\mathsf{syn}}$ calculates the end-to-end delay kt as the upper bound delay threshold that prevents forks during the block propagation delay. For that purpose, $\mathcal{F}_{\mathsf{syn}}$ queries each $n_i \in N_r$ after kt. For the nodes that report $b_{r+1} \preceq b_r$ as the latest block on C, $\mathcal{F}_{\mathsf{syn}}$ puts them in N_s as the synchronized nodes; otherwise in N_u as the non-synchronized nodes, where $N_r = N_s \cup N_u$. $\mathcal{F}_{\mathsf{syn}}$ then computes $\mathcal{N}_{\mathsf{syn}}$ as the ratio $|N_s|/|N_r|$. If $\mathcal{N}_{\mathsf{syn}} > 0.5$ (an honest majority is synchronized), $\mathcal{F}_{\mathsf{syn}}$ notifies each $n_i \in N_r$ that the network is synchronized.

Figure 1: Ideal functionality for the Bitcoin network synchronization. The two conditions specified in the ideal functionality ensure that all *reachable* nodes in Bitcoin eventually receive a block and the maximum block propagation delay among the *reachable* nodes is bounded by a delay threshold parameter to prevent forks with a high probability.

Network Synchronization. Mining nodes relay their blocks to other mining and non-mining nodes to facilitate the network synchronization. In Bitcoin, the network synchronization determines how many nodes have a *correct* and *up-to-date* blockchain at any time [22]. When a block is relayed in the network, it is desirable for all nodes to receive the block at the same time to avoid mining power waste and forks [22]. However, since the Bitcoin network consists of ≈6-9K *reachable* nodes with an average outdegree of 10 [7], it is improbable for a block to be relayed to all the nodes instantly. Therefore, non-uniform block propagation affects the network synchronization, which can be exploited to create forks and reduce the cost of a majority attack [19, 34].

It is essential to understand the role of *reachable* nodes in provisioning the network synchronization. For instance, when a mining node mines a block, it relays that block to all the *reachable* and *unreachable* nodes connected to the mining node. When a *reachable* node receives the block, it relays that block to other *reachable* and *unreachable* nodes connected to it. However, when an *unreachable* node receives the block, it can only relay that block to a *reachable* node (since no two *unreachable* nodes can be directly connected). As such, if all *unreachable* nodes are partitioned from the network, blocks can still reach all the *reachable* nodes, allowing them to synchronize over the blockchain. In contrast, if all *reachable* nodes are partitioned from the network, blocks cannot propagate in the P2P network, thus preventing the network synchronization. In other words, the Bitcoin blockchain synchronization strongly relies on the number of *reachable* nodes in the network and their network topology. Mining pools own both *reachable* and *unreachable* nodes, and use *reachable* nodes to propagate their blocks [37]. In Figure 18 (Appendix F), we illustrate the anatomy of the Bitcoin P2P network, highlighting the unique roles different node types.

3 IDEAL FUNCTIONALITY FOR BITCOIN NETWORK SYNCHRONIZATION

The attack proposed in this work is motivated by the discrepancies between the Bitcoin's ideal synchronization model and its real world behavior. However, as mentioned in §1, the existing security modes do not fully incorporate the network synchronization in the Bitcoin security model, particularly based on the current Bitcoin Core rules. Therefore, we first propose the ideal functionality for Bitcoin network synchronization, which we then contrast with the real world behavior to construct SyncAttack.

For the ideal functionality, we assume a set of *reachable* nodes N_r as the "Interactive Turning Machine"s (ITM) that execute the Nakamoto consensus for l rounds, arbitrated by a trusted party $\mathcal{F}_{\mathsf{syn}}$. Each $n_i \in N_r$ establishes ten outgoing connections, making the average network outdegree $\deg^+(N_r)$, which then enables the block propagation in $k = \log_{\deg^+(N_r)}|N_r|$ steps. Each step adds a fixed delay, t, and the network synchronizes if no fork appears in time kt while N_r maintains a minimum outdegree $\deg^+_{\min}(N_r)$. Figure 1 provides the ideal functionality details, and Theorem 3.1

specifies bounds on $\deg^+(N_r)$ and block propagation delay that preserve the network synchronization.

Theorem 3.1 (Ideal Functionality $\mathcal{F}_{\mathsf{SYN}}$). *By maintaining both (1) a minimum outdegree $\left(deg^+(N_r) \geq deg^+_{min}(N_r)\right)$ and (2) an upper bound block propagation delay threshold ($kt \leq T$), $\mathcal{F}_{syn}$ guarantees synchronization with a high probability.*

In Appendix §A, we prove Theorem 3.1 and provide the lower bound for $\deg^+(N_r)$ and the upper bound for kt based on the Bitcoin protocol specifications. Compared to the existing theoretical frameworks [22, 35], we make the following refinements in our ideal functionality to correctly model the network synchronization based on the rules encoded in Bitcoin Core.

(1) We acknowledge the default outgoing connection limits for a *reachable* node in Bitcoin by setting $\deg^+(N_r)$ =10. In the prior theoretical models [22], the authors assume that the overlay is strongly connected which leads to a synchronous communication. However, the assumption of a strongly connected topology undermines the $\deg^+_{\min}(N_r)$ requirement that the real world network must satisfy. Therefore, our ideal functionality captures the correct state of the overlay topology. **(2)** We note that forks that violate the blockchain *consistency* are not solely determined by the adversary's mining power. Instead, if any of the two conditions in Theorem 3.1 are violated, forks will appear even in the absence of an adversary. **(3)** Since the network synchronization depends on $\deg^+_{\min}(N_r)$ and kt, we side-step the mining power distribution in our ideal functionality. We assume a uniform hash rate distribution among the *reachable* nodes, which enables us to analyze the fork probability irrespective of the biased mining power distribution. In other words, Figure 1 is a lower bound construction that incorporates synchronization in the primordial Bitcoin design proposed in [31].[4]

In summary, our ideal functionality embraces the reality of the real world overlay topology by incorporating the network synchronization in the Bitcoin security model. As such, by violating Theorem 3.1, an adversary can deteriorate the network synchronization to violate the blockchain *consistency* property through forks. In the following section, we present measurements to analyze how closely the real world network follows the ideal behavior.

Analysis Notations. In addition to the analysis notations defined in §3, in the following we provide the notations that will be used in the rest of the paper. We define $\widehat{f_h}(x)$ as the kernel density estimation, N_r as the *reachable* nodes, N_i as the newly arriving nodes, N_e as the departing nodes, R_p as the persistent nodes, M_r as the *reachable* mining nodes, M_i as the newly arriving mining nodes, M_e as the existing mining nodes, $\mathcal{A}$ as an adversary, A_r as the adversary's *reachable* nodes, and A_u as the adversary's *unreachable* nodes. We also define O as the nodes to which we connect in the testnet experiment (§5.3), and P as the connections established to the source node S_N in that experiment.

4 BITCOIN NETWORK MEASUREMENTS

In this section, we present measurements of the real world characteristics of the Bitcoin network. We focus our study on (1) the network synchronization in the real world, (2) blockchain forks and the network outdegree due to variations in the network synchronization, (3) the network churn caused by the *permissionless* nature, and (4) partitioning possibilities due to the churn.

For measurements, we collected data from an online service called Bitnodes that connects to all Bitcoin *reachable* nodes and reports their latest blockchain view [8]. We collected Bitnodes data from January 01, 2021 to March 01, 2021.[5]

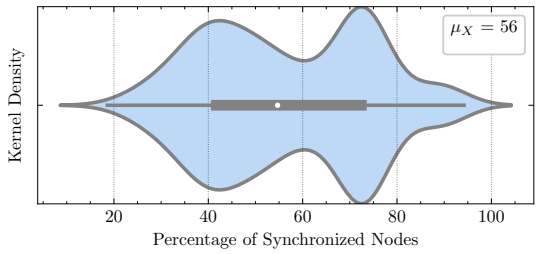

Figure 2: Network synchronization results obtained by applying Heuristic 1 on our dataset. On average, only 56% *reachable* nodes have an up-to-date blockchain.

4.1 Bitcoin Network Synchronization

To measure and analyze the network synchronization, we compare the latest block reported by Bitnodes with the latest block on the blockchain tip of all the *reachable* nodes. Since Bitnodes crawlers are connected to all the *reachable* nodes [8, 38], they instantly receive a newly mined block from any *reachable* node. As such, Bitnodes' view of the Bitcoin network is similar to the view of $\mathcal{F}_{\mathsf{syn}}$ in Figure 1. Taking that into account, we assume Bitnodes as $\mathcal{F}_{\mathsf{syn}}$ and apply **Heuristic 1** to analyze the network synchronization.

Heuristic 1. *When $\mathcal{F}_{syn}$ receives $b_{r+1} \preceq b_r$ from any $n_i \in N_r$, $\mathcal{F}_{syn}$ invokes* **Evaluate** *in* Figure 1 *and counts the percentage of $|N_r|$ that report b_{r+1} on C.*

After applying **Heuristic 1** on our dataset, we obtain the set of synchronized nodes $N_s \subset N_r$. We then sample $\mathcal{N}_{\mathsf{syn}}$ =100×$|Ns|/|N_r|$ as a list $X = (x_1, x_2, ..., x_z)$, where $x_i \in X$ is the percentage value of $\mathcal{N}_{\mathsf{syn}}$ for each block and z is the total number of blocks. Next, we calculate the kernel density estimation $\widehat{f_h}(x)$ of X [43] using the following formula.

$$\widehat{f_h}(x) = \frac{1}{z}\sum_{i=1}^{z} K_h\,(x - x_i) = \frac{1}{mh}\sum_{i=1}^{z} K\left(\frac{x - x_i}{h}\right) \qquad (1)$$

In (1), K is the Gaussian kernel and h is the kernel bandwidth applied using Scott's rule [39]. In Figure 2, we plot $\widehat{f_h}(x)$ against X, showing that the average network synchronization is 56%, which is marginally above the threshold specified in the ideal functionality (Figure 1). In other words, on average, only 56% of the nodes connected to Bitnodes report an up-to-date blockchain.

In ideal conditions, the network synchronization should be close to 100% so that all nodes synchronize and share the same blockchain

[4]The lower bound construction in Figure 1 can be easily extended to accommodate for the biased distribution of the mining power in the current Bitcoin network. However, the resulting model must satisfy Theorem 3.1 in order to ensure that all the *reachable* nodes eventually synchronize over the blockchain.

[5]Bitnodes provides an API through which network snapshots can be collected. Each snapshot provides the latest Bitcoin block and the latest block reported by all the *reachable* nodes connected to Bitnodes' crawlers.

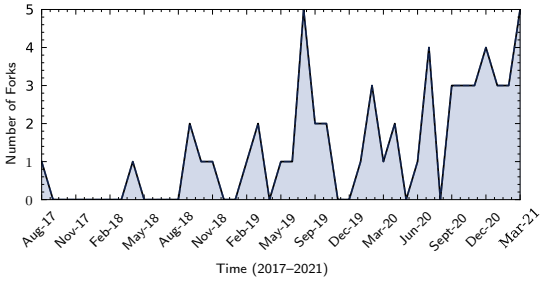

Figure 3: Number of forks reported in the Bitcoin network between 2017 and 2021. The number of forks is increasing each year with up to 22 forks reported in 2020. Since January 2021, the Bitcoin blockchain has already forked 12 times.

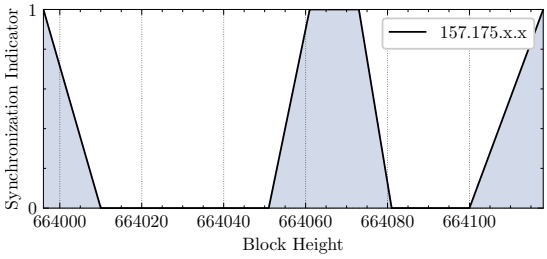

Figure 4: Network synchronization pattern of a node obtained from algorithm 2. When the node was synchronized, the corresponding value in the list was marked 1 (synchronization indicator). Therefore, the shaded region shows all the blocks for which the node remained synchronized.

view. In 2012, the Bitcoin network synchronization was strong and 90% of the *reachable* nodes received a block within 12 seconds [19]. However, in the last few years, the network synchronization appears to be deteriorating [33] per the data reported by Bitnodes. Moreover, we did not find any value of $x_i \in X$, where $\mathcal{N}_{\text{syn}}$ was 100%, and the maximum and minimum values for $\mathcal{N}_{\text{syn}}$ were 86.3% and 15.7%, respectively. [6]

4.1.1 Bitcoin Forks. Since the Bitcoin network synchronization has been deteriorating in the last few years [33, 38**?**], it is logical to assume that the network has a high orphaned block rate due to forks. Although forks can be found at online block explorer services such as *Bitcoin.com* [14], most explorer services have stopped reporting recent forks. An alternative method of determining forks is to run a full node, execute the *getchaintips* command, and count all the branches forked from the main chain. Currently, two online services (*ChainQuery* [10] and *ForkMonitor* [16]) report blockchain forks using data from their full nodes.

We crawled data from *ChainQuery* [10] and observed that the deteriorating synchronization has indeed led to an increase in the number of blockchain forks. In Figure 3, we report the total number of forks from 2017 to 2021. Based on the data available at *ChainQuery*, in 2017, only one fork was reported in the network. In 2018, the number of forks increased and the blockchain forked four times during the year as the network synchronization became weaker [38]. In 2019, the number of forks significantly increased to 14, with five forks reported in August 2019. In 2020, the blockchain forked 22 times, with more than two forks reported each month (except May 2020). From January 2021 to March 2021, the blockchain forked 12 times, with five forks reported in March 2021. Since May 2020, each fork results in a loss of more than 6.25 bitcoins to the miner whose orphaned block is not included in the blockchain. Therefore, in 2021 alone, miners have lost more than $4 million due to forks.

From results in Figure 3, it is fair to assume that the network synchronization impacts the number of blockchain forks in the Bitcoin network. The occurrence of forks also indicates that the mining nodes do not form an exclusive community in the overlay topology (*i.e.,* all mining nodes are directly connected). Our conclusions are supported by the prior work conducted by Miller *et al.* [20], who observed that mining nodes do not have a high network outdegree and they typically follow the standard node configurations (*i.e.,* default incoming and outgoing connections). As a result, when the network synchronization deteriorates, the mining nodes are also likely to suffer from it, leading to an increase in forks [19].

4.1.2 Network Outdegree. Besides the overall synchronization of 56%, another key observation in Figure 2 is the non-uniform width of $\widehat{f_h}(x)$ that indicates variations in $\deg^+(N_r)$ (*i.e.,* the changing network outdegree resulting in varying block propagation patterns). It is therefore worth investigating if the network outdegree falls below the minimum outdegree ($\deg^+(N_r) \leq \deg^+_{\min}(N_r)$), thus preventing block delivery to a group of nodes for a long time. A test case to determine this condition would be to find a non-synchronized node at a particular blockchain height and observe the node's synchronization pattern for all subsequent blocks. If the node stays behind the blockchain for all subsequent blocks, we can conclude that $\deg^+(N_r) \leq \deg^+_{\min}(N_r)$, and there is no path in the overlay network that delivers blocks to that node.

In Algorithm 2, we present our technique to determine if $\deg^+(N_r)$ is below $\deg^+_{\min}(N_r)$ for a long time. We initialize an object O_{syn}, with $n_i \in N_r$ as the object keys and an empty list as the value for each key. We then iterate over each block $c_j \in C$ and add 1 to the list if the latest block reported by n_i is equal to c_j (*i.e.,* the node is synchronized), and 0 otherwise. Finally, we return O_{syn} from algorithm 2, and apply **Heuristic 2** to determine if $\deg^+(N_r)$ is below $\deg^+_{\min}(N_r)$.

Heuristic 2. *For all the list values corresponding to a key in O, if there is a value* 1, *after any sequence of* 0's , *then* $\deg^+(N_r)$ is eventually greater than $\deg^+_{\min}(N_r)$.

Heuristic 2 specifies that if a node was behind the chain in the past and eventually caught up, then there exists a path in the overlay network that delivers blocks to that node. Therefore, the average network outdegree is greater than the minimum outdegree.

After applying algorithm 2 on our dataset, we did not find any *reachable* node that stayed behind the blockchain indefinitely. As an example, in Figure 4, we plot the synchronization pattern of a *reachable* node for 120 consecutive blocks. We mask the last two

[6] Among the open source tools available to monitor the network synchronization, we report data obtained from Bitnodes, which is widely used in the existing research for measuring and mapping the Bitcoin network. Assuming that the results reported by Bitnodes can have discrepancies (*i.e.,* due to protocol implementation or network sampling time), it highlights that measuring the network synchronization remains largely an open question. We note, however, that our proposed attack in §5 is largely unaffected by the synchronization values reported by Bitnodes.

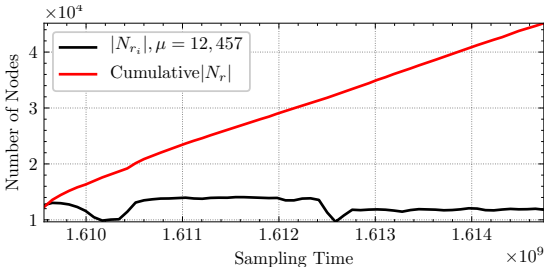

Figure 5: The cumulative and the average number of *reachable* nodes present in the Bitcoin network at any time. The gap between the two lines indicates a notable network churn caused by the *permissionless* network.

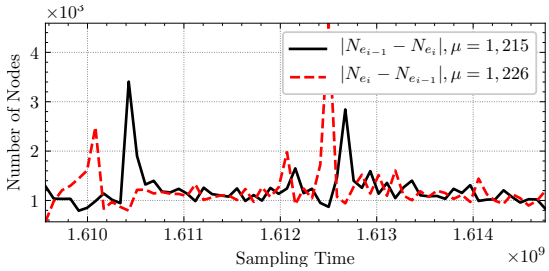

Figure 6: The number of arriving and departing nodes in the Bitcoin network. On average, in 60 days, 1,226 nodes joined and 1,215 nodes departed from the network every day.

octets of the node's IP address to preserve its privacy. Figure 4 shows that each time the node was behind the blockchain, it eventually caught up and synchronized on the latest block. The node's behavior in Figure 4 validates that there exists a path in the overlay topology through which the node eventually receives the blocks.

Figure 4 also shows variations in the synchronization pattern, indicating that the block reception depends on the node's location in the overlay network relative to the mining nodes. For instance, for blocks 664060 and 664080, the node was found to be synchronized, suggesting a close proximity with the mining nodes of those blocks. On the other hand, for blocks 664020 and 664040, the node was distant from the mining nodes of those blocks, and therefore did not receive blocks even after a long time.

These observations lead to two possible characterizations of the network outdegree. (1) The network outdegree is always greater than the minimum outdegree and the lack of synchronization is likely due to the node's location in the overlay topology. (2) In the worst case assumption, even if the network outdegree becomes less than the minimum outdegree, it eventually recovers since the non-synchronized nodes eventually catch up with the blockchain.

4.1.3 Key Takeaways. From the synchronization analysis of the real world network, we make the following key conclusions. First, the overall network synchronization is not ideal since only 56% nodes report an up-to-date blockchain at any time. As a result, the number of blockchain forks has increased in the last few months. Second, despite imperfect synchronization, the network outdegree is still greater than the minimum outdegree, which results in a quick fork resolution and further enables the non-synchronized nodes to catch up with the blockchain. Since the observed forks resolve quickly, the blockchain *consistency* property is not violated. In the following, we show that an adversary can prevent the fork resolution by exploiting the Bitcoin network's *permissionless* nature.

4.2 Bitcoin Network Churn

Since the Bitcoin network is *permissionless*, nodes can join or leave the network at any time [31]. The arrival and departure of nodes create the churn and changes the network outdegree, which subsequently affects the block propagation and the network synchronization. In SyncAttack, the adversary exploits the churn to partition the existing nodes and the arriving nodes, and uses that partitioning to create forks and violate the blockchain *consistency*. In this section, we analyze the Bitcoin network churn to extract useful insights for the SyncAttack construction.

4.2.1 Measurement Results. **Network Size.** In 60 days, we collected 45,165 IP addresses of *reachable* nodes with ≈12,457 *reachable* nodes present in the network at any time. Figure 5 shows the number of *reachable* nodes ($|N_r|$) present in the network at any time, as well as the cumulative number of unique IP addresses of the *reachable* nodes collected during the measurements. The gap between the two lines in Figure 5 indicates a notable churn rate since the number of unique IP addresses increased continuously while the number of *reachable* nodes present in the network at any time remained almost constant (≈12.5K).

Arriving and Departing Nodes. After observing the network churn, we analyze the vulnerable network state created by the churn. When a *reachable* node departs, all its connections are dropped, including the incoming connections from its peers. Those peers then try new outgoing connections to complete their default outgoing slots (10 in Bitcoin including the two *feeler* connections). If no other *reachable* node accepts their connection requests, their average network outdegree decreases, affecting the network synchronization (Figure 1 and §A).

Similarly, if a node joins the network and no *reachable* node accepts its connections, the network outdegree remains low. Furthermore, if an adversary occupies all the node's incoming and outgoing connections, the node can be partitioned from the rest of the network (*i.e.,* eclipse attack) [26]. Therefore, the node arrivals and departures can create an imbalance in the overall network outdegree, which can be exploited by an adversary to split the network and control the communication model.

To analyze the number of arriving and departing nodes, we denote $N_{e_{i-1}} - N_{e_i}$ as the set of nodes present in the previous day $i-1$, but absent in the current day i. The resulting value $|N_{e_{i-1}} - N_{e_i}|$ gives the number of nodes that departed from the network on day i. Conversely, $|N_{e_i} - N_{e_{i-1}}|$ gives the number of arriving nodes that were not found on the previous day. In Figure 6, we plot the number of departing nodes $|N_{e_{i-1}} - N_{e_i}|$ and the number of arriving nodes $|N_{e_i} - N_{e_{i-1}}|$ for 60 days. The result shows that, on average, 1,215 nodes departed from the network and 1,226 new nodes joined the network every day. While overall the network size remained stable

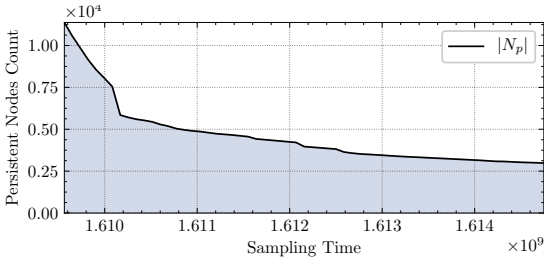

Figure 7: The number of persistent nodes R_p in the Bitcoin network. Note that over time, the curve flattens and we find 2,984 nodes that stayed persistently in the network.

during this period, the network experienced a notable churn rate and a high variation in the Bitcoin network outdegree, leading to a varying network synchronization observed in Figure 2.

It is important to note that if an adversary strategically occupies all the incoming connection slots of the existing nodes in the network, a newly arriving node will not be able to establish an outgoing connection to the existing nodes. Moreover, if the adversary occupies the incoming and outgoing connections of the arriving nodes, then the network will be partitioned between arriving nodes and existing nodes. Due to the churn, the number of existing nodes will decrease and the number of arriving nodes will increase over time, allowing an adversary to orchestrate a mining race between the two groups. This form of network partitioning is at the core of the SyncAttack, and in §5, we will show how the adversary achieves this without mining power.

Persistent Nodes. Our measurements also revealed that despite the network churn, 2,984 nodes did not leave the network during the entire measurement study. For simplicity, we call them "persistent nodes" (R_p), and plot them in Figure 7 by counting the common elements in N_{e_i} and $N_{e_{i-1}}$. The key feature of the persistent nodes is that, unlike the arriving nodes, the outgoing connections of persistent nodes cannot be easily manipulated by the adversary [25, 26]. For instance, if a node $n_i \in R_p$ establishes all its outgoing connections to other nodes in R_p, those connections will not drop despite the departure of other nodes that experience the churn.

Mining Nodes. Our next step is to study the characteristics of the mining nodes and analyze how they can be partitioned by the adversary. Unfortunately, Bitnodes dataset does not reveal the mining nodes. In practice, an adversary can simply connect to all the *reachable* nodes and identify the ones that relay the blocks before other *reachable* nodes in the network. Those nodes can be labeled as the mining nodes, and the adversary can determine their network lifetime and churn. Prior works [20, 37] that used this mining nodes detection technique have reported that the mining nodes have a long network lifetime. Naïvely extrapolating their findings to our measurements conducted in 2021, we can assume that mining nodes could be among the persistent nodes. As mentioned in the previous section, attacking persistent nodes is harder than attacking nodes that experience the churn.

Although the above-mentioned assumption seems logical, it may not be entirely true based on the differences in the experiment duration in our study and prior works. For mining nodes detection, [20, 37] conducted measurements for a few weeks and found that mining nodes do not experience significant churn. In contrast, our measurements in Figure 7 span two months, which means that mining nodes could have persistently stayed in the network for one month and may have departed later on. Based on these observations, we can make the following deductions about the mining nodes, their network communication model, and feasibility of SyncAttack in each communication model.

(1) Mining nodes are among the persistent nodes and they only establish outgoing connections to each other (*i.e.,* form a community). In such a case, SyncAttack will be infeasible since the adversary cannot easily manipulate the outgoing connections among the mining nodes. **(2)** Mining nodes are not among the persistent nodes and they exhibit the churn. In such a case, SyncAttack will be highly feasible since the adversary can partition the mining nodes and control their communication by exploiting the churn. **(3)** Some mining nodes are among the persistent nodes while others experience the churn. Moreover, some mining nodes do not only establish outgoing connections to the other mining nodes. In such a case, SyncAttack will still be feasible since the adversary can partition the network between the new and the existing mining nodes.

We now evaluate how closely the real world Bitcoin communication model maps onto each of the communication model described above. In the first model, if mining nodes are among the persistent nodes and make direct outgoing connections with each other then they must instantly receive blocks from each other. As a result, the blockchain should not fork as frequently as shown in Figure 3. Therefore, we can rule out the possibility that if mining nodes are among the persistent nodes then they only establish direct connections to each other. The prior work by Miller *et al.* [20] also showed that the mining nodes follow a standard topology without forming an exclusive community.

The second model is also less likely because a few mining pools such as AntPool and SlushPool have been part of the Bitcoin network for many years, and they have consistently mined blocks [9]. The churn among the nodes owned by AntPool means that a mining pool replaces an old node with a new node to perform block relaying. Setting up a new node and synchronizing with the blockchain can take a long time, which would be costly for the mining pool. Therefore, some mining nodes are likely among the persistent nodes.

The third model appears to be the most plausible characterization of the mining nodes in the current Bitcoin network. As mentioned in §1, between 2020 and 2021, several new mining pools (*i.e.,* ArkPool, SBI Crypto, and Foundry USA) joined the network, while other mining pools (*i.e.,* PHash.io, NovaBlock, and BytePool) left the network. Therefore, despite a few mining nodes being persistently present in the network, there is still some churn caused by the newly arriving and departing mining nodes.

In order to empirically evaluate the third communication model among the mining nodes, we conducted a follow-up experiment by deploying a *supernode* in the Bitcoin network and connecting to the *reachable* nodes. We then adopted the mining nodes detection technique specified in [6, 20] and discovered the IP addresses of 790 mining nodes over a duration of ≈37 days (5439 blocks). Our results confirmed that the mining nodes follow the third communication model whereby some mining nodes persistently stay in the network while others experience the churn. More precisely, we found that on average, there were ≈384 mining nodes present in the network at any time with at least six mining nodes persistently staying in the

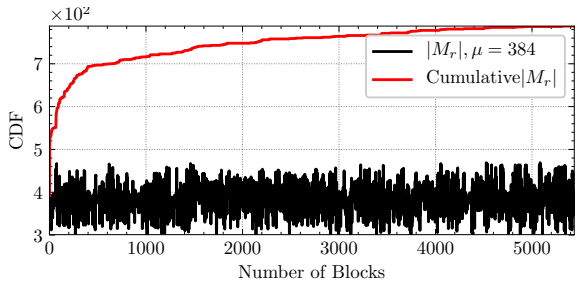

Figure 8: The number of mining nodes ($|M_r|$) present in the network at any time, and the cumulative number of the mining nodes discovered during the measurement study. The gap between the two lines demonstrates the churn among the mining nodes.

network while others experiencing churns. In Figure 8, we plot the cumulative number of unique mining nodes and the total number of mining nodes present in the network at any time. The gap between the two lines shows the churn among the mining nodes.

4.2.2 Key Takeaways. From the churn analysis, we have the following observations. The Bitcoin network has a notable churn rate and ≈10% *reachable* nodes depart from the network every day, replaced by almost an equal number of arriving nodes.[7] The churn also provides clues about the synchronization pattern observed in Figure 2. When nodes leave the network, the network outdegree decreases, which is then improved by the arriving nodes. However, the arriving nodes are usually behind the blockchain and it takes time to synchronize with the network. As a result, they are often behind the blockchain when the Bitnodes service queries them. This indicates that the churn is another key factor behind the synchronization pattern observed in Figure 2.

5 THE SYNCATTACK

We now present the SyncAttack by colliding the network synchronization with the *permissionless* nature of the Bitcoin network. At a high level, in SyncAttack, an adversary occupies all the incoming connections of the existing nodes, and the incoming and outgoing connections of the arriving nodes. As a result, the arriving nodes–including the mining nodes–cannot establish connections with the existing nodes, creating a network partition controlled by the adversary. As the number of the existing and arriving mining nodes changes due to the network churn, the mining power splits between the two partitions, breaking the synchronization and creating forks. The adversary exploits those forks to violate the blockchain *consistency* and double-spend without using any mining power. In this section, we present the SyncAttack threat model, followed by the attack procedure.

5.1 Threat Model

For the SyncAttack threat model, we use the formalism introduced in §3 by specifying N_r *reachable* nodes, with each $n_i \in N_r$ establishing O_i=10 outgoing connections and accepting I_i=115 incoming connections. We further divide N_r into N_i arriving nodes and N_e existing nodes. Prior to the attack, $|N_i|$=0 and $|N_e|$=12.5K (Figure 5).[8]

Next, we assume an adversary $\mathcal{A}$ who runs two sets of *reachable* nodes (A_r and A_u). Each $a_i \in A_r$ maintains a Bitcoin blockchain with modified source code to allow more than 115 incoming connections from N_r[9]. Furthermore, each $a_i \in A_r$ establishes 1 outgoing connection to every node in A_u. Similarly, each $a_i \in A_u$ also accepts more than 115 incoming connections and establishes more than 12.5K outgoing connections to $n_i \in N_r$. However, no $a_i \in A_u$ maintains a blockchain, and instead executes a lightweight script with the following functions. **(1)** Establishes an outgoing connection to $n_i \in N_r$ by performing the TCP handshake and exchanging the `VER` and `VERACK` messages [42]. **(2)** In response to `GETADDR` message, only relays the IP addresses of A_r or A_u to $n_i \in N_r$ in all the `ADDR` messages. **(3)** Selectively relays new transactions and blocks to $n_i \in N_r$, received from any $a_i \in A_r$ or $a_i \in A_u$. **(4)** Optionally requests old blocks from $n_i \in N_r$ and discards them upon reception.

The above functions allow $a_i \in A_u$ to mimic a *reachable* node's behavior without maintaining the ≈330GB blockchain. By only relaying A_r and A_u in the `ADDR` messages, $\mathcal{A}$ ensures that the IP addresses of its *reachable* nodes reach the `new` table of each $n_i \in N_r$.

Number of Nodes required for SyncAttack. As stated earlier, $\mathcal{A}$ aims to occupy all the incoming and outgoing connections of N_i, and all the incoming connections of N_e. Therefore, it is important to estimate the number of the nodes required to achieve this objective in order to determine the attack feasibility.

Since the nodes A_r maintain the blockchain and allow incoming connections, only $|A_r|$=10 will be sufficient to occupy O_i=10 slots of all the *reachable* nodes. On the other hand, occupying I_i=115 incoming connection slots of any $n_i \in N_r$ will require $\mathcal{A}$ to host up to $|A_u|$=115 nodes. From each $a_i \in A_u$, $\mathcal{A}$ will establish 12.5K outgoing connections in order to occupy all the incoming connection slots of N_r. Given that there are 65K available ports per host, $\mathcal{A}$ can easily establish 12.5K connections from a single machine.

5.2 Attack Procedure

At a high level, the attack procedure involves $\mathcal{A}$ carrying out the following set of activities.

(1) $\mathcal{A}$ calculates the total number of *reachable* slots of the existing *reachable* nodes that are not occupied by the *reachable* and *unreachable* nodes. **(2)** $\mathcal{A}$ strategically occupies those slots (using $a_i \in A_u$) so that no *reachable* connection slot is available to support any new connections between two *reachable* nodes. **(3)** $\mathcal{A}$ ensures that all the connections established by $a_i \in A_u$ are not evicted when any *reachable* or *unreachable* node tries to replace them.

In summary, $\mathcal{A}$ occupies all the available slots of the existing *reachable* nodes and prevents any new connection between two *reachable* nodes. For that purpose, $\mathcal{A}$ also needs to account for the *node eviction logic* [12] that can potentially replace $\mathcal{A}$'s connections

[7] It is possible that nodes switch their IP addresses. However, that behavior is similar to the departure and arrival of nodes since all the incoming and outgoing connections may change after switching the IP address.

[8] $|N_e|$=12.5K is an optimistic estimate based on Figure 5. The number of *reachable* nodes can often vary (*i.e.,* less than 7.6K in September, 2020 [8]).

[9] $\mathcal{A}$ modifies the *net.h* file [13] to increase the number of incoming connections.

Algorithm 1: Occupying All Incoming Connections

Input: N_r, A_u
foreach $n_i \in N_r$ **do**
  Initialize: C_s= 0, E_s= 0
  /* Occupy all incoming connections of n_i using 1 IP address and a unique port */
  while n_i accepts connections **do**
    select one a_j and connect to n_i using a different port
    increment C_s
  Allow 16 connections in A_u to relay blocks to n_i and 4 connections in A_u to relay transactions received from other a_i in A_r or A_u
  /* Replace and evict up to 101 existing connections */
  while n_i accepts connections **do**
    select a_j with a unique network group and connect to n_i
    increment E_s
  Return E_s, and C_s
Return R_a=0

to the *reachable* nodes. The current eviction policy specifies that if a node has all its incoming connection slots full and it receives a new incoming connection request, then the node can potentially evict one of the existing connections to allow the new incoming connection. During the eviction process, the node first preserves 28 connections that it considers to be useful (*i.e.,* connections that relay new blocks). [10] Among the remaining 87 connections, half are protected based on the longest uptime. The protected connections can include up to 50% localhost connections that may not have the longest up-time. Finally, among the remaining connections selected for eviction, the network group (determined by prefixes) with the most incoming connections is chosen and the node with the latest connection time is evicted [12].

In order to not be evicted, $\mathcal{A}$ selects one node in A_u and uses it to establish 16 connections to each $n_i \in N_r$[11]. Each connection uses the same IP address and a different port. However, through those 16 connections, $\mathcal{A}$ constantly relays new transactions and blocks to each $n_i \in N_r$, thus preventing the adversary's connections from being evicted. For the remaining 99 connections, $\mathcal{A}$ uses a unique network group for each connection. In case any of the 99 connections has a lower ping time or where the existing connections of $n_i \in N_r$ belong to the same network group, the adversary's connections will evict the existing connections. By following this procedure, $\mathcal{A}$ (1) occupies all the available connection slots of each $n_i \in N_r$, (2) possibly evicts the existing connections between the *reachable* nodes, and (3) prevents from being evicted by the new incoming connections.

To formally analyze the attack procedure, we define $R_c = N_r \times 115$ as the total number of slots available for the *reachable* and *unreachable* nodes to occupy. Among those slots, we assume that R_o slots are already occupied by those nodes prior to the attack. Accordingly, we define $R_a = R_c - R_o$ as the available slots that $\mathcal{A}$ occupies using lightweight scripts. When R_a=0, there is no available slot in the network for any new *reachable* or *unreachable* node. This is the focal point of the SyncAttack, since $\mathcal{A}$ causes a denial-of-service by ensuring that *no reachable node accepts any new incoming connection from other nodes in the network.* In algorithm 1, we describe how $\mathcal{A}$ occupies all the available slots in the network.

Algorithm 1 shows that for each $n_i \in N_r$, $\mathcal{A}$ first establishes multiple incoming connections from the same IP address and different ports. For each successful connection, $\mathcal{A}$ counts the occupied connection slots C_i. Once n_i stops accepting the incoming connections, $\mathcal{A}$ selects 16 of its established connections to relay new transactions and blocks to n_i[12]. Next, $\mathcal{A}$ replaces all its other connections by using 99 IP addresses from A_u. For each successful connection, $\mathcal{A}$ counts the evicted connections E_i. If the difference between E_i and C_i is more than 16, then $\mathcal{A}$ also evicts connections between honest nodes (*i.e.,* due to a low ping time or multiple honest connections from the same network group)[13]. In [1], we show how an adversary uses a lightweight script to occupy the incoming connection slots of a node. We conducted the experiment on our own *reachable* node, occupying only the available connection slots without affecting any existing connections between our node and other honest nodes. Due to ethical concerns, we did not conduct the second phase of the experiment where the adversary evicts connections among honest nodes (algorithm 1).

When algorithm 1 completes, $\mathcal{A}$ ensures that the available connection slots R_a=0, and no $n_i \in N_r$ can establish any outgoing connection to any other $n_j \in N_r$. However, $n_i \in N_r$ can establish an outgoing connection to any *reachable* node controlled by $\mathcal{A}$, since those nodes still accept incoming connections. Once R_a=0 and the churn occurs, $\mathcal{A}$ starts to control the links between nodes in $|N_i|$ and $|N_e|$ to violate the ideal functionality specifications. In the following, we show $\mathcal{A}$'s strategies during the network churn.

5.2.1 Arriving Nodes. When a new node n_i joins the network for the first time, it queries a list of DNS seeds hardcoded in the *chainparams.cpp* file [11]. The DNS query returns a list of *reachable* addresses to which n_i establishes outgoing connections. After successfully connecting to a *reachable* node, n_i sends the `GETADDR` message to that node in order to receive an `ADDR` message containing up to 1000 IP addresses of other Bitcoin nodes.

Since R_a=0 after algorithm 1 is executed, n_i can only establish an outgoing connection if the DNS seeds return an IP address of any node in A_i or A_u. Once n_i connects to any node in A_r or A_u, n_i only receives the IP addresses of A_r and A_u in the `ADDR` message. As a result n_i establishes all 10 outgoing connections to the *reachable* nodes controlled by the adversary.

When $\mathcal{A}$ learns the IP address of n_i, $\mathcal{A}$ runs algorithm 1 to occupy a_i's incoming connections through nodes in A_u. If n_i accepts incoming connections, algorithm 1 will ensure that all its incoming slots are occupied and no other node can connect to n_i. As a result, all I_i and O_i of n_i are occupied by $\mathcal{A}$.

A constraint in this attack procedure is that the DNS seeds must relay at least one IP address in A_r or A_u to n_i. To analyze how this can be achieved, we explored the DNS seed specification provided by a Bitcoin Core developer, which states that the DNS seeders *"return a good sample"* of *reachable* nodes in their response [18]. This

[10] Preserved connections include: 4 connections from random network groups (determined by their prefixes), 8 connections with a minimum ping time, 4 connections that relayed new transaction, and up to 12 connections that relayed recent blocks [12].
[11] Bitcoin Core allows multiple incoming connections from the same IP address.

[12] Since $\mathcal{A}$ is connected to all *reachable* nodes through A_u, $\mathcal{A}$ can instantly receive new transactions and blocks from $n_i \in N_r$, which can be used to prevent the eviction of 16 connections in A_u that use the same IP address and a different port.
[13] This is a natural caveat of the node eviction policy. The connections among the honest nodes can also be evicted by the adversary.

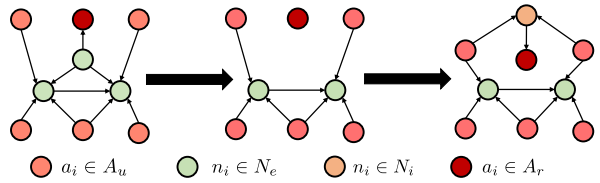

Figure 9: SyncAttack illustration showing how $\mathcal{A}$ occupies all the connections of the arriving nodes N_i and the outgoing slots of N_e, left opened by the departure of an existing node.

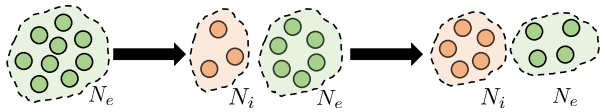

Figure 10: Changes in N_e and N_i along the network churn. Over time, the size of N_e decreases, the size of N_i increases.

means that the Bitcoin nodes owned by the DNS seed providers know a few IP addresses of *reachable* nodes in the network. Since $\mathcal{A}$'s nodes are connected to all the *reachable* nodes, $\mathcal{A}$ can send `ADDR` messages to its connections containing only the IP addresses in A_r and A_u. This procedure will increase the probability of IP addresses in A_r and A_u being relayed by the DNS seeders.

5.2.2 Departing Nodes. When a *reachable* node departs from the network, its *reachable* connections will have one less outgoing connection that they have established with the departing node. When R_a=0, those nodes are unable to connect to any other node in N_r. However, if they have an IP address of any node in A_r or A_u in their `new` or `tried` tables, they eventually establish an outgoing connection to complete their outgoing slots.

If the departing node rejoins the network at any time (with the same IP address or a new IP address), the node skips the DNS querying phase and attempts connections from its `new` and `tried` tables. If 11 seconds elapse without a successful connection, the node queries the DNS seeders [11]. Considering the attack procedure described in Algorithm 1, $\mathcal{A}$ occupies the connection slots of the *reachable* nodes to which the node was connected prior to its departure from the network. Therefore, upon rejoining the network, the node cannot connect to its previously connected nodes or any other *reachable* node due the unavailability of connection slots (R_a=0). As a result, the node eventually connects to the *reachable* nodes (A_r) controlled by the adversary since they accept the incoming connection requests. Subsequently, all the incoming connection slots of the node are occupied by A_u, allowing $\mathcal{A}$ to control the incoming and outgoing connection slots of the node. Therefore, rejoining the network with same IP address or a new IP address does not prevent the node from partitioning.

5.2.3 Network Partitioning. By maintaining R_a=0, $\mathcal{A}$ ensures that all the incoming and outgoing connections of N_i are established with A_u and A_r, respectively. Moreover, when any $n_i \in N_e$ departs from the network, its *reachable* connections only connect with A_r or A_u. Figure 9 illustrates the state of the network when a node departs and a new node joins the network. Since no node in N_i can connect to any node in N_e (R_a=0), the network is partitioned between N_i and N_e. Moreover, the size of $|N_e|$ decreases and the size of $|N_i|$ increases with the churn, as shown in Figure 10. From Figure 7, we note that it takes ≈60 days to flatten the curve, from which we obtained $|\mathrm{R}_p|$=2,984 nodes. Therefore, the size $|N_i|$ will become $|N_e|-|\mathrm{R}_p|$ in 60 days.

5.2.4 Communication Model. We now examine the communication model of the network N_i under the churn and evaluate its compliance with the ideal functionality specifications in Figure 1.

Since $\mathcal{A}$ controls all the incoming and outgoing connections of each $n_i \in N_i$, $\deg^+(N_i)$ becomes 0 (*i.e.,* no edge between the honest nodes). This allows $\mathcal{A}$ to violate the first condition in Theorem 3.1, since $\deg^+(N_i)$ remains 0 despite the increasing network size. Additionally, by controlling all the connections in N_i, $\mathcal{A}$ can delay the block propagation in N_i by more than kt seconds, violating the second condition in Theorem 3.1. This shows that when algorithm 1 is followed by the churn, the network is partitioned and $\mathcal{A}$ can completely deteriorate the network synchronization in N_i.

Given the fact that several new mining pools joined the network in 2021 [17], their mining nodes therefore become part of N_i with the incoming and outgoing connections occupied by $\mathcal{A}$. $\mathcal{A}$ can then orchestrate a block race between mining nodes in N_i and N_e to fork the chain and successfully double-spend. In the following, we show how $\mathcal{A}$ double-spends without using any mining power.

5.2.5 Double-spending in SyncAttack. We propose two constructions for SyncAttack in which $\mathcal{A}$ exploits the churn to double-spend a transaction. The first construction is based on the primitive Bitcoin model proposed by Nakamoto [31], and we show how the *permissionless* settings can be exploited to violate the blockchain *consistency*. Our generalized construction can be easily extended to other PoW-based cryptocurrencies as long as they experience network churn and $\mathcal{A}$ can feasibly deteriorate the network synchronization. Our second construction is specifically tailored to the current Bitcoin network settings (*i.e.,* acknowledging the biased distribution of the mining power). Due to the space limitations, we present the generalized construction in Appendix §B and here we present the Bitcoin-specific construction of the SyncAttack.

In Figure 11, we demonstrate how $\mathcal{A}$ launches the SyncAttack to double-spend in the current Bitcoin network. We categorize the mining nodes into two groups, namely $M_i \in N_i$ and $M_e \in N_e$. First, $\mathcal{A}$ identifies M_e by annotating the nodes that produce new blocks [20]. $\mathcal{A}$ then waits for the churn to find some of the existing mining nodes in M_e to be replaced by the new mining nodes in M_i. Next, $\mathcal{A}$ generates a transaction tx and a conflicting transaction tx' using the same "Unspent Transaction Output" (UTXO). $\mathcal{A}$ relays tx to M_i and a user A, and tx' to M_e and another user B.

$\mathcal{A}$ then stops relaying blocks between M_i and M_e, thereby forcing them to mine two branches of the chain ($C_1 \leftarrow M_i$ and $C_2 \leftarrow M_e$). Moreover, $\mathcal{A}$ relays C_1 to user A and C_2 to user B. Eventually, when both C_1 and C_2 acquire a $k-$confirmations promised by $\mathcal{A}$ to both A and B, $\mathcal{A}$ receives products from both users and releases the longest branch to diffuse the fork. In Figure 11, we assume that $\alpha < \beta$, thus C_2 is longer than C_1 and tx is rejected. As a result, user

Double-spending in the SyncAttack

Input: Mining nodes $M_i \in N_i$, $M_e \in N_e$, and adversary $\mathcal{A}$. $\mathcal{A}$ detects all the mining nodes in M_e by monitoring the nodes that release a new block [20, 37]. Initially, M_i=0 and each $m_i \in M_e$ mines on C.

Churn: As the size of N_i increases with the churn, $\mathcal{A}$ detects new mining nodes in M_i when they release blocks.

Attack Initiation: $\mathcal{A}$ starts the attack by preventing block exchanges between M_e and M_i. When any $m_i \in M_i$ produces a new block, $\mathcal{A}$ only relays that block to other mining nodes in M_i. Similarly, when $m_i \in M_e$ mines a new block, the block is only relayed to other mining nodes in M_e (either by $\mathcal{A}$ or other nodes in N_e whose connections are not controlled by $\mathcal{A}$). As a result, the hash rate splits into $M_i \leftarrow \alpha$ and $M_e \leftarrow \beta$, where α is the hash rate of new mining nodes and β is the hash rate of existing mining nodes.

Issue Double-spent Transactions: $\mathcal{A}$ selects two users A and B with non-mining nodes $n_a \in N_i$ and $n_b \in N_e$, respectively. $\mathcal{A}$ then generates a transaction tx and a double-spent transaction tx$'$ from the same UTXO [31]. For tx, $\mathcal{A}$ selects A as the recipient, and for tx$'$, $\mathcal{A}$ selects B as the recipient. For each transaction, $\mathcal{A}$ sets a high mining fee and sends tx to M_i and tx$'$ to M_e.

Block Race: Assuming b_r as the latest block on C, when the block race starts, the mining nodes in M_i mine $b_{r+1} \preceq b_r$, while the mining nodes in M_j mine $b'_{r+1} \preceq b_r$. The blockchain C splits into $C_1 \preceq C$ and $C_2 \preceq C$. The block b_{r+1} contains t_x and the block b'_{r+1} contains tx'. Upon receiving b_{r+1} and b'_{r+1}, $\mathcal{A}$ relays b_{r+1} to M_i and n_a, and b'_{r+1} to M_j and n_b.

Receiving Product: When both branches (C_1 and C_2) become k blocks (typically $k = 6$ is the confirmation factor in Bitcoin [11]), both A and B will deliver the product to $\mathcal{A}$ or further spend tx and tx$'$ with other users.

Dissolving Fork: The rate at which C_1 and C_1 will grow depends on the distribution of α and β. For simplicity, we assume $\beta > \alpha$, and C_2 will extend faster than C_1. Once $\mathcal{A}$ receives a product from both A and B, $\mathcal{A}$ releases the longer chain C_2 to all the *reachable* nodes N_r in the Bitcoin network. Complying with the longest chain rule [22, 31], all mining and non-mining nodes switch to C_2 and discard C_1. $\mathcal{A}$ double-spends since tx is invalidated.

Figure 11: Double-spending in the SyncAttack where $\mathcal{A}$ orchestrates mining on two blockchain branches and generates conflicting transactions on each branch. When $\mathcal{A}$ receives the reward for each transaction, $\mathcal{A}$ releases the longest branch to diffuse the fork. Note that despite diffusing the fork, $\mathcal{A}$ still controls N_i and can always re-launch the attack.

A is tricked and $\mathcal{A}$ double-spends without using any mining power, thereby violating the blockchain *consistency*.

It can be argued that miners can exchange blocks outside the Bitcoin network via direct communication channels [15], in which case they can detect an attack in the *reachable* network. However, even in that case, the SyncAttack can succeed since not all mining nodes communicate with each other over some external channels [20]. $\mathcal{A}$ can easily detect covert communication among miners by observing the difference in blocks relayed to them by $\mathcal{A}$ and the blocks that they mine and relay back to $\mathcal{A}$. If the two blocks are inconsistent, $\mathcal{A}$ infers that the miners are covertly communicating outside the Bitcoin P2P network. If $\mathcal{A}$ detects a covert communication among the miners, it can simply discard their blocks and orchestrate a block race only among the mining nodes that use the P2P network to relay their blocks.

It is worth mentioning that the SyncAttack can work in any distribution of the mining power. For instance, as long as only one new mining node joins the network ($M_i \neq 0$), $\mathcal{A}$ can isolate it from the rest of the network to successfully double-spend. Therefore, the minimum attack duration is subject to the arrival of a new mining node and a block race of at least six blocks. Based on the results shown in Figure 8, new mining nodes can be observed in the network frequently (sometimes within a day), and a block race of six blocks takes ≈ 60 minutes. Therefore, a double-spend attack targeting a single mining node can be launched within a day. However, if the adversary wants to double-spend as well as waste the maximum hashing power of the Bitcoin network, the adversary might have to wait for ≈60 days (§5.2.3) in order to completely control the communication model among all major mining nodes. The attack construction presented in Figure 11 is modeled on an adversary that aims to double-spend and waste the maximum hashing power. The proposed construction can also help in evaluating the blockchain *safety* and *liveness* properties using theoretical frameworks presented in [22, 34].

Attack Cost. In SyncAttack, $\mathcal{A}$ hosts $|A_r|$=10 *reachable* nodes with unique IP addresses and over 500GB storage space for blockchain. Moreover, $\mathcal{A}$ also hosts $|A_u|$=115 *reachable* nodes that only execute lightweight scripts. Among those 115 *reachable* nodes, 16 are emulated using the same IP address and different ports. The remaining 99 are hosted in 99 network groups with unique IP addresses. $\mathcal{A}$ can host $|A_r|$=10 as cloud instances. Currently, multiple cloud services (*i.e.,* DigitalOcean and Amazon) support instance hosting across multiple network groups and geographical locations. The estimated 60 days cost for $|A_r|$=10 is ≈$1,800 [4]. For $|A_u|$=100 (16 connections with the same IP address and 99 with unique IP addresses), $\mathcal{A}$ can simply acquire a static IP address and host $|A_u|$ on Docker containers across different cloud services. With a modest estimate of ≈$23 for acquiring an IP address [2] and $5 for hosting a virtual machine with Docker containers [32], the cost of operating $|A_u|$=100 nodes is ≈$2,800. Therefore, the total attack cost is ≈$4,600.

It is important to note that SyncAttack is significantly cheaper than prior IP address based partitioning attacks (*i.e.,* Eclipse attack [26]). A recent work [42] showed that eclipsing a Bitcoin node requires an adversary to own more than 100K IP addresses. The total cost of acquiring 100K IP addresses by an adversary can be prohibitively high (≈$2.3 million using the calculations provided

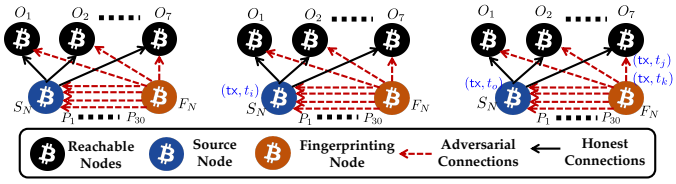

Figure 12: Experiment setup for transaction deanonymization. The source node S_N and the fingerprinting node F_N were connected to (O=O_1 … O_7). Additionally, F_N established 30 connections to S_N (1 IP and 30 ports).

above and [2]). In comparison, the total cost for SyncAttack is only ≈$4,600, making it more cost effective.

5.3 SyncAttack: Associated Risks

Although our main focus in this work is the practical analysis of Bitcoin network synchronization and SyncAttack, during our investigation we also discovered some other related weaknesses in Bitcoin Core that have critical security and privacy implications.

A key weakness that we discovered is that Bitcoin Core allows multiple incoming connections from the same IP address, and an adversary can exploit this weakness to (1) link transactions to an IP address, and (2) exhaust a victim node's bandwidth to reduce its hashing power. Due to the page limits, we have moved the detailed analysis of transaction deanonymization and bandwidth exhaustion to Appendix §C and Appendix §D. Below, we only provide a brief summary of our experiment on transaction deanonymization.

In 2015, Bitcoin replaced the transaction relaying protocol from *trickle spreading* to *diffusion spreading* to prevent an adversary from connecting to all nodes and link transactions to IP addresses (see Appendix §C for a detailed background on *trickle spreading* and *diffusion spreading*). We hypothesized that by establishing multiple connections to the same node (ideally through the same computer), the anonymity guarantees of *diffusion spreading* can be weakened.

Experiment and Results. We set up a source node S_N in the testnet. S_N established outgoing connections to seven *reachable* nodes (O=O_1, … O_7), and we executed *getpeerinfo* RPC command to obtain their IP addresses. Next, we deployed our fingerprinting node F_N that established one outgoing connection to each node in O, and 30 connections (P = P_1…P_{30}) to the source node S_N. All 30 connections were established using the same IP address and a different port. The experiment setup is shown in Figure 12. From S_N, we generated 19 transactions that were relayed to O and F_N, following the *diffusion spreading* protocol. Additionally, F_N received 15 transactions from S_N and O, that were not generated by S_N.

We then followed a heuristic (formally described in §C) that if S_N is the transaction source, then F_N receives the transaction from S_N (through one of the 30 connections), before receiving that transaction from O. Similarly, if S_N is not the transaction source, then F_N receives that transaction from O before receiving that transaction from S_N. Among the total (19+15=34) transactions, 27 transactions satisfied the heuristic, yielding ≈79.4% accuracy.[14]

[14] Among the 19 transactions generated by S_N, 13 were received by F_N before O.

6 SYNCATTACK UNDERWAY AND COUNTERMEASURES

6.1 Ongoing Activities

Considering the feasibility of SyncAttack and other associated risks, it is logical to assume that malicious nodes could be exploiting the network weaknesses in order to deteriorate synchronization or to perform other malicious activities. To investigate such activities, we set up a *reachable* node and observed the number of incoming connections that use the same IP address. We conducted our experiment for three days and analyzed the IP addresses of the incoming connections using the *getpeerinfo* RPC API.

In Figure 19 (Appendix F), we report our results showing instances where our node received up to 33 incoming connections from the same IP address. Figure 19 also shows two time windows spanning ≈21 hours and ≈4.5 hours, in which the number of incoming connections significantly increased. Given that each Bitcoin node randomly selects an IP address for the outgoing connection, it is improbable that up to 33 *unreachable* nodes selected the same IP address to request the blockchain. Therefore, the two anomalies observed in Figure 19 suggest likely malicious activities. Although the activities observed at our node do not represent all the security risks associated with the SyncAttack, they however clearly indicate actions that can lead to the deterioration of synchronization. In order to extrapolate the impact of such activities, in §D, we conduct experiments to demonstrate how establishing multiple connections and requesting bulk data can impact a miner's hash rate.

6.2 SyncAttack Countermeasures

In this section, we present the SyncAttack countermeasures.

[C1] Preventing Multiple Connections. Our analysis in §5.1 and §5.3 shows that by allowing multiple incoming connections from the same node, an adversary (1) lowers the requirement for SyncAttack from 115 IP addresses to 100 IP addresses, (2) deanonymizes transactions with a high accuracy, and (3) reduces the hash rate of the mining nodes. Therefore, the first line of defense to mitigate deanonymization and increase the SyncAttack cost is to prevent multiple incoming connections from the same IP address. The modified client version with **[C1]** can be found in [1].

[C2] Limiting Multiple Connections. When we shared **[C1]** with the developers community, their feedback suggested that while **[C1]** is effective, it might affect the *unreachable* nodes behind NAT. By limiting 1 IP address per incoming connection, two nodes behind NAT may not be able to connect to the same set of *reachable* nodes. Therefore, the next step was to find a balance between the number of nodes that can be allowed from the same IP address. For that purpose, it is important to determine the total number of *unreachable* nodes behind NAT in the Bitcoin network. In 2020, we had set up our nodes that connected to the *reachable* nodes and iteratively sent `GETADDR` messages to collect the IP addresses known to the *reachable* nodes. We used that dataset to find the right balance between the number of nodes that can be allowed from the same IP address. Through our dataset and Bitnodes [8], we found that there were up to ≈8K–12K *reachable* nodes in the network at any time. Moreover, by analyzing the `GETADDR` messages, we found that there were ≈195K *unreachable* nodes (details in Figure 20).

Among those *unreachable* nodes, we found that ≈12K addresses were behind NAT (using the same IP address and a different port). Diving into the number of NATed nodes (12K) by the conservative estimate of *reachable* nodes (8K), we found that a *reachable* node accommodates up to ≈2 nodes behind the NAT. As a result, up to ≈2 nodes behind the NAT can connect to the same *reachable* node, which is an optimal number to support that NATed nodes without significantly sacrificing the benefits achieved through **[C1]**.

[C3] Improving the Eviction Policy. As long as there is network churn, SyncAttack is possible. However, SyncAttack can be mitigated if nodes in N_i evict the adversary's connections in N_e. We note that the current eviction policy favors nodes that have longest connection time. Since nodes in N_i join the network after the adversary has already occupied the connection slots in N_e, therefore, N_i cannot evict those connections. To prevent the adversary's hegemony in N_e, we propose a change to the current connection eviction policy by randomizing the victim. For example, when any $n_i \in N_i$ attempts a connection with $n_j \in N_e$, n_j can select two random network groups and (1) evict one connection with the oldest connection time, and (2) evict one connection with latest connection time. As a result, the adversary will lose one connection with n_j, breaking the partitioning between N_i and N_e, and mitigating the SyncAttack.

To summarize, **[C2]** is a more suitable countermeasure to mitigate the SyncAttack. However, it does not prevent transaction deanonymization. In contrast, **[C1]** increases the cost for SyncAttack and largely preserves the transaction anonymity as well as the effective hash rate. Finally, **[C2]** is the optimal implementation of **[C1]** that marginally decreases the SyncAttack cost without impacting the *unreachable* nodes. A combination of **[C1]** and **[C3]** could be useful for the mining nodes as it would prevent SyncAttack and preserve the hash rate. On the other hand, a combination of **[C2]** and **[C3]** could be useful for non-mining nodes as it would mitigate SyncAttack while also supporting *unreachable* nodes.

7 RELATED WORK

In this section, we review the prior works on Bitcoin security that are related to our study in this paper.

Theoretical Models. The two notable theoretical models that characterize the Bitcoin security properties are (1) the *lock-step* synchronous model by Garay *et al.* [22], and (2) the *non-lock-step* synchronous model by Pass *et al.* [34]. Although both models assume well-connected topology, the *non-lock-step* synchronous model incorporates a block propagation delay of ≈10 seconds between the mining nodes. Especially different from their model specifications, our ideal functionality (Figure 1) presents a more realistic model consistent with the current Bitcoin deployment.

Network Synchronization Measurements. In 2013, Decker *et al.* [19] conducted the first measurement study to analyze block propagation in the Bitcoin network. They observed that a Bitcoin block takes ≈12 seconds to reach 90% *reachable* nodes. In 2018, deterioration in blockchain synchronization started to occur and it was reported in [38]. However, both studies did not take into account the *permissionless* nature of the Bitcoin network, and the attacks that they proposed required using the mining power.

Partitioning Attacks. In 2017, Apostolaki *et al.* [6] analyzed the Bitcoin network hash rate distribution across the Internet and observed that an adversary can reduce the hash rate by more than 60% by hijacking only 3 ISPs. Although they did not discuss the effects of such an attack on the blockchain *consistency*, it can be easily inferred that the attack would reduce the majority attack requirement to only 21% of the hash rate.

In 2020, Tran *et al.* [42] proposed a stealthier attack that partitions the Bitcoin network without manipulating the routing paths. Their attack relied on poisoning the `new` and `tried` tables of the victim node so that the node establishes all its outbound connections to the adversary. As discussed in §5, the attack requires an adversary to own thousands of IP addresses, which can only be mounted by a strong adversary with a budget to afford thousands of IP addresses.

In 2015, Heilman *et al.* [26] identified a vulnerability in Bitcoin Core that allowed an adversary to eclipse the victim node by occupying all the victim's incoming and outgoing connections. However, the vulnerability has since been patched in Bitcoin Core. Nevertheless, the cost for an Eclipse attack was higher than SyncAttack, since the adversary was expected to own thousands of IP addresses.

SyncAttack: A Comparative Evaluation. In terms of the attack cost and attack feasibility, SyncAttack is less costly than all the existing partitioning attacks in the literature. The attack feasibility is self-evident since we consider a much weaker adversary than prior works [6, 37, 42]. Similarly, SyncAttack is also cost effective since the adversary benefits from (1) the natural partitioning created by churn, and (2) the weaknesses in the existing network. Alongside the core contributions that are foundational to the Bitcoin design, our work also exposes risks associated with the discovered weaknesses (*i.e.,* transaction deanonymization and hashrate reduction). We minimize those risks by proposing refinements to Bitcoin client.

8 CONCLUSION

In this paper, we have investigated the network synchronization and incorporated it into the Bitcoin security model. Our measurements and analysis present a contrast between the ideal functionality and the real world network behavior to expose various attack vectors that can be exploited to deteriorate the network synchronization and violate the Bitcoin blockchain *consistency* property. Especially new to the Bitcoin security model is our observation that the network churn can be exploited to partition the network and deteriorate the network synchronization. We formally analyze the churn-based partitioning by presenting SyncAttack that allows an adversary to double-spend without using any mining power. Moreover, we identify weaknesses in the current Bitcoin deployment that can be exploited to lower the SyncAttack cost, deanonymize transactions, and reduce the *effective* network hash rate. Accordingly, we propose three refinements in Bitcoin Core as countermeasures for SyncAttack.

Acknowledgements. This work is supported in part by NRF grant 2016K1A1A2912757. We want to thank Alin Tomescu for shepherding our paper. We also would like to thank Ben Price, Ameer Sheikh, and Ashar Ahmad for helping with the supernode deployment. S. Chen was supported in part by the NSF grant CNS-2007153 and a Commonwealth Cyber Initiative grant.

REFERENCES

[1] 2021. Bitcoin Synchronization Attack. (2021). https://anonymous.4open.science/r/106b2297-8daf-4b75-a209-6468a8dc91c1/

[2] 2021. IP Address Marketplace: Worldwide. (Jan 2021). https://ipv4marketgroup.com/ipv4-pricing/

[3] 2021. PoW 51% Attack Cost. (2021). https://www.crypto51.app/

[4] Amazon. 2021. EC2 Instance Types & Pricing. https://ec2pricing.net/. (2021).

[5] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. Feb 2019. SABRE: Protecting Bitcoin against Routing Attacks. In *Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA*. https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/

[6] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy*. 375–392. https://doi.org/10.1109/SP.2017.29 https://doi.org/10.1109/SP.2017.29.

[7] Annika Baumann, Benjamin Fabian, and Matthias Lischke. 2014. Exploring the Bitcoin Network. In *International Conference on Web Information Systems and Technologies*, Valérie Monfort and Karl-Heinz Krempels (Eds.). 369–374. https://doi.org/10.5220/0004937303690374

[8] B.Community. 2021. Bitnodes: Discovering All Reachable Nodes In Bitcoin. (2021). https://bitnodes.earn.com/

[9] Blockchain. 2018. Hashrate Distribution. (2018). https://blockchain.info/pools.

[10] ChaniQuery. 2021. bitcoin-cli getchaintips – ChainQuery. https://chainquery.com/bitcoin-cli/getchaintips. (2021).

[11] Bitcoin Community. 2018. Bitcoin Core version history. (2018). https://github.com/bitcoin/bitcoin.

[12] Bitcoin Community. 2020. Bitcoin Net.cpp File Containing Node Eviction Logic. https://github.com/bitcoin/bitcoin/blob/master/src/net.cpp. (2020).

[13] Bitcoin Community. 2020. bitcoin/net.h at master · bitcoin/bitcoin. https://github.com/bitcoin/bitcoin/blob/master/src/net.h. (2020).

[14] Bitcoin Community. 2021. Bitcoin (BTC) Block Explorer. https://explorer.bitcoin.com/btc. (2021).

[15] Bitcoin Community. 2021. Bitcoin Relay Network. https://bit.ly/3tjtAUa. (2021). (Accessed on 01/29/2021).

[16] Bitcoin Community. 2021. Fork Monitor. https://forkmonitor.info/stale/btc/677102. (2021).

[17] Bitcoin Community. 2021. Pool Stats - BTC.com. https://btc.com/stats/pool?pool_mode=all. (2021).

[18] Christian Decker. 2021. (2021). https://bitcoinstats.com/network/dns-servers/

[19] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE International Conference on Peer-to-Peer Computing*. 1–10. https://doi.org/10.1109/P2P.2013.6688704 https://doi.org/10.1109/P2P.2013.6688704.

[20] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. 2019. TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions. In *International Conference on Financial Cryptography and Data Security (Lecture Notes in Computer Science)*, Vol. 11598. Springer, 550–566. https://doi.org/10.1007/978-3-030-32101-7_32

[21] Giulia C. Fanti and Pramod Viswanath. Dec 2017. Deanonymization in the Bitcoin P2P Network. In *Annual Conference on Neural Information Processing Systems 2017 Long Beach, CA, USA*. 1364–1373. https://tinyurl.com/y72zgvtk.

[22] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *International Cryptology Conference on Advances in Cryptology*. 291–323. https://doi.org/10.1007/978-3-319-63688-7_10

[23] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. 2015. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *ACM SIGSAC Conference on Computer and Communications Security*. 692–705. https://doi.org/10.1145/2810103.2813655

[24] Cyril Grunspan and Ricardo Pérez-Marco. 2017. Double spend races. *CoRR* abs/1702.02867 (2017). arXiv:1702.02867 http://arxiv.org/abs/1702.02867

[25] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In *ISOC Network and Distributed System Security Symposium*. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/ndss201701-3HeilmanPaper.pdf.

[26] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *USENIX Security Symposium*. 129–144. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[27] Marco Alberto Javarone and Craig Steven Wright. 2018. Modeling a Double-Spending Detection System for the Bitcoin Network. *CoRR* abs/1809.07678 (2018). arXiv:1809.07678 http://arxiv.org/abs/1809.07678

[28] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *USENIX Security Symposium*. 279–296. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias

[29] Jing Li and Dongning Guo. 2020. Liveness and Consistency of Bitcoin and Prism Blockchains: The Non-lockstep Synchronous Case. In *IEEE International Conference on Blockchain and Cryptocurrency*. https://doi.org/10.1109/ICBC48266.2020.9169464

[30] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin Lightweight Client Privacy using Trusted Execution. In *USENIX Security Symposium*. 783–800. https://www.usenix.org/conference/usenixsecurity19/presentation/matetic

[31] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). https://bitcoin.org/bitcoin.pdf.

[32] Digital Ocean. 2021. Spin up your virtual machine in just 55 seconds. https://try.digitalocean.com/. (2021). (Accessed on 01/27/2021).

[33] Giuseppe Pappalardo, Tiziana di Matteo, Guido Caldarelli, and Tomaso Aste. 2018. Blockchain inefficiency in the Bitcoin peers network. *EPJ Data Sci.* 7, 1 (2018), 30. https://doi.org/10.1140/epjds/s13688-018-0159-3

[34] Rafael Pass, Lior Seeman, and Abhi Shelat. 2016. Analysis of the Blockchain Protocol in Asynchronous Networks. *IACR Cryptology ePrint Archive* 2016 (2016), 454. http://eprint.iacr.org/2016/454

[35] Ling Ren. 2019. Analysis of Nakamoto Consensus. Cryptology ePrint Archive, Report 2019/943. (2019). https://eprint.iacr.org/2019/943.

[36] Meni Rosenfeld. 2014. Analysis of Hashrate-Based Double Spending. *CoRR* abs/1402.2009 (2014). arXiv:1402.2009 http://arxiv.org/abs/1402.2009

[37] Muhammad Saad, Afsah Anwar, Srivatsan Ravi, and David A. Mohaisen. Nov 2021. Revisiting Nakamoto Consensus in Asynchronous Networks: A Comprehensive Analysis of Bitcoin Safety and Chain Quality. (Nov 2021).

[38] Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, and Aziz Mohaisen. 2019. Partitioning Attacks on Bitcoin: Colliding Space, Time, and Logic. In *IEEE International Conference on Distributed Computing Systems*. 1175–1187. https://doi.org/10.1109/ICDCS.2019.00119

[39] David W. Scott. 1992. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley. https://doi.org/10.1002/9780470316849

[40] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. 2019. Performance Modeling and Analysis of the Bitcoin Inventory Protocol. In *IEEE International Conference on Decentralized Applications and Infrastructures*. 79–88. https://doi.org/10.1109/DAPPCON.2019.00019

[41] Michael Bedford Taylor. 2017. The Evolution of Bitcoin Hardware. *Computer* 50, 9 (2017), 58–66. https://doi.org/10.1109/MC.2017.3571056

[42] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. 2020. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In *IEEE Symposium on Security and Privacy*. 894–909. https://doi.org/10.1109/SP40000.2020.00027

[43] Peng Wang, Hua Deng, Yi Min Wang, Yue Liu, and Yi Zhang. 2020. Kernel Density Estimation Based Gaussian and Non-Gaussian Random Vibration Data Induction for High-Speed Train Equipment. *IEEE Access* 8 (2020), 90914–90923. https://doi.org/10.1109/ACCESS.2020.2994224

APPENDIX

A IDEAL FUNCTIONALITY PROOF

In this section, we provide the proof sketch for Theorem 3.1.

Proof. For the proof sketch, we show that the protocol in Figure 1 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{syn}}$ by modelling the real world network characteristics [8]. For that purpose, we set up the model parameters for each condition in Theorem 3.1, and use values from the real world Bitcoin network [8].

The first condition in Theorem 3.1 $\left(\deg^+(N_r) \geq \deg^+_{\min}(N_r)\right)$ refers to the Bitcoin network's capability of delivering blocks to all the *reachable* nodes. Therefore, $\deg^+_{\min}(N_r)$ characterizes the minimum number of edges required to construct a connected overlay topology among the *reachable* nodes. Logically, if the network outdegree falls below the minimum outdegree $\left(\deg^+(N_r) < \deg^+_{\min}(N_r)\right)$, a group of *reachable* nodes will not be connected to the network, thus weakening the network synchronization [40].

To show that our ideal functionality satisfies the first condition in Theorem 3.1, we derive the minimum Bitcoin network outdegree from [40], and compare it with the empirical values from the real world Bitcoin network [8]. From [40], we note that among $|N_r|$

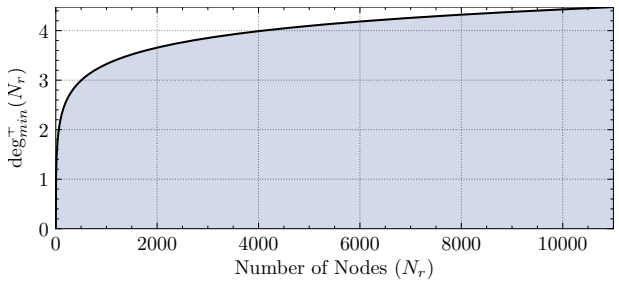

Figure 13: Relationship between the network size $|N_r|$ and the minimum outdegree $\deg^{+}_{min}(N_r)$ required for a connected topology. In the current network size of ≈12.5K nodes [8], $\deg^{+}_{min}(N_r)$ must be greater than 4.1.

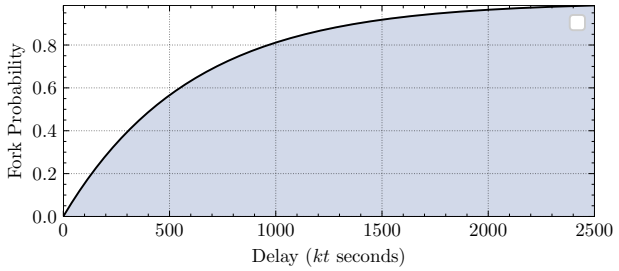

Figure 14: Probability of fork due to block propagation delay kt. We note that at kt=2500 seconds, the fork probability becomes greater than 0.99. Therefore, we set our upper bound delay threshold T=2500 seconds.

reachable nodes, the minimum outdegree $\deg^{+}_{min}(N_r)$ is bounded by the following relationship.

$$\deg^{+}_{min}(N_r) \geq \left\lceil \frac{|N_r|}{|N_r|-1} \log_{O_i}(|N_r|) \right\rceil \tag{2}$$

Using (2), we plot $\deg^{+}_{min}(N_r)$ against $|N_r|$ in Figure 13. We increase $|N_r|$ from 0 to 11K nodes, which is currently the number of *reachable* nodes in the network [8]. Figure 13 shows that among $|N_r|$=11K nodes, if $\deg^{+}_{min}(N_r)$ is greater that 4.1 (*i.e.,* 5), then there is a path from each node to every other node to deliver a block. Furthermore, through source code inspection, we observe that each *reachable* node in the real world network establishes ten outgoing connections (O_i=10), making that the network outdegree ($\deg^{+}(N_r) = 10$) [11]. Since $\deg^{+}_{min}(N_r) < \deg^{+}(N_r)$, therefore, our ideal functionality satisfies the first condition in Theorem 3.1 by correctly modeling the synchronization requirement.

The second condition in Theorem 3.1 ($kt \leq T$) refers to the Bitcoin network's capability of preventing forks during block propagation. For that purpose, we specify that if the end-to-end block propagation delay kt is below the delay threshold parameter T, the probability of a fork remains 0.99, thus preserving the blockchain *consistency* properties [22, 28].

In order to obtain a realistic value for T, we identify events during block propagation that can cause forks. Consider a node n_0 that mines a block $b_{r+1} \preceq b_r$ at time t_a. Next, consider another node $n_{|N_r|}$ as the last node in N_r to receive $b_{r+1} \preceq b_r$ at t_b. Therefore, the end-to-end delay kt becomes $t_b - t_a$, and a fork appears if $n_{|N_r|}$ mines $b'_{r+1} \preceq b_r$ between $t_b - t_a$. Let $\mathbb{P}[X = \mathbf{F}]$ be the probability that a fork appears during block propagation time kt. From [19], we note that $\mathbb{P}[X = \mathbf{F}]$ can be calculated as follows.

$$\mathbb{P}[X = \mathbf{F}] = 1 - (1 - \lambda)^{kt} \tag{3}$$

In (3), λ is the probability of finding a block in 1 second. In Bitcoin, λ=1/600, where 600 is the average block time. Using λ=1/600, (3) can also be written as follows.

$$\mathbb{P}[X = \mathbf{F}] = 1 - \left(1 - \frac{1}{600}\right)^{kt} \tag{4}$$

In Figure 14, we plot (4) by varying kt from 0 to 2500 seconds, and observe that $\mathbb{P}[X = \mathbf{F}]$ increases with kt. Since we aim to maintain $\mathbb{P}[X = \mathbf{F}]$ 0.99, we derive the cutoff value, T=2500 seconds, which limits $\mathbb{P}[X = \mathbf{F}]$ 0.99. Moreover, given that $\deg^{+}(N_r)$=10, we calculate the propagation delay t in each step k as $t = T/\log_{\deg^{+}(N_r)} |N_r| \approx 32$ seconds. Our bound on T is realistic since prior measurements reported $kt \approx 12$ seconds in Bitcoin [19].

To conclude, our ideal functionality for the Bitcoin network synchronization is admissible in the Bitcoin computation model since we show that (1) the average network outdegree is greater the minimum required outdegree, and (2) the realistic bound of $kt \leq 2500$ prevents forks with a high probability. □

Forks can occur due to any of the two conditions mentioned in Theorem 3.1. As such, if forks do not resolve for k consecutive blocks either due to (1) continuously high block propagation delay, or (2) $\deg^{+}(N_r)$ remaining below $\deg^{+}_{\min}(N_r)$, then the Bitcoin blockchain will violate the common prefix and chain quality properties that capture *consistency* and *liveness* [22, 34].

Our ideal functionality in Figure 1 is based on the Bitcoin primordial design in [31] which assumed a democratized network in which 1 CPU had 1 Vote. If Bitcoin were to follow the primordial design, then the SyncAttack adversary would be required to orchestrate a mining race between N_i and R_p (§5.2). However, due to mining centralization, the current Bitcoin network has significantly departed from [31]. So we had to tailor our attack construction by modeling a mining race between the mining nodes only (§5.2). To the adversary's advantage, since all mining nodes experience churn, therefore, the SyncAttack is more feasible in practice. Moreover, we want to emphasize that our ideal functionality and attack can be easily generalized across other PoW-based blockchain systems that are inspired from Bitcoin.

B GENERALIZED CONSTRUCTION FOR SYNCATTACK

In Figure 15, we present the generalized construction of SyncAttack for the Nakamoto consensus. We model our construction of the Bitcoin primordial design [31] by assuming a uniform distribution of the mining power. In the generalized attack construction, $\mathcal{A}$ waits for six days until $|N_i| == |N_e|$. As a result, in addition to violating

Double-spending in the SyncAttack

Input: N_i, N_e, and adversary $\mathcal{A}$. Initially, N_i=0, N_e=12.5K, and each $n_i \in N_e$ mines on C. We assume 1CPU=1 Vote (*i.e.,* a uniform hash rate distribution as envisioned in [31]).

Churn: $\mathcal{A}$ waits for 8 days until the size of N_i is equal to the size of N_e. During the network churn, whenever any $n_i \in N_e$ or $n_j \in N_e$ produces a block, $\mathcal{A}$ relays that block to all nodes N_r through A_r. As a result, all nodes have the same ledger C on their blockchain.

Attack Initiation: When the size of N_i is equal to the size of N_e, $\mathcal{A}$ stops relaying blocks between N_i and N_e to split the network. When any $n_i \in N_i$ produces a new block, $\mathcal{A}$ only relays that block to other nodes in N_i. Similarly, when $n_j \in N_e$ mines a new block, the block is only relayed to other nodes in N_e (either by $\mathcal{A}$ or other nodes in N_e whose connections are not controlled by $\mathcal{A}$). As a result, the hash rate splits into $N_i \leftarrow \alpha$ and $N_e \leftarrow \beta$, where both α and β are 0.5.

Issue Double-spent Transactions: $\mathcal{A}$ selects two users A and B with nodes $n_a \in N_i$ and $n_b \in N_e$, respectively. $\mathcal{A}$ then generates a transaction tx, and a double-spent transaction tx$'$ from the same UTXO [31]. For tx, $\mathcal{A}$ selects A as the recipient, and for tx$'$, $\mathcal{A}$ selects B as the recipient. For each transaction, $\mathcal{A}$ sets a high mining fee and sends tx to N_i and tx$'$ to N_e.

Block Race: Assuming b_r to be the latest block on C, when the block race starts, the mining nodes in N_i mine $b_{r+1} \preceq b_r$, while the mining nodes in N_e mine $b'_{r+1} \preceq b_r$. The blockchain C splits into $C_1 \preceq C$ and $C_2 \preceq C$. The block b_{r+1} contains t_x and the block b'_{r+1} contains tx'. Upon receiving b_{r+1} and b'_{r+1}, $\mathcal{A}$ relays b_{r+1} to N_i and b'_{r+1} to N_e. Additionally, $\mathcal{A}$ relays b_{r+1} to n_a and b'_{r+1} to n_b.

Receiving Product: When both branches (C_1 and C_2) become k blocks (typically k = 6 is the confirmation factor in Bitcoin [11]), both A and B will deliver the product to $\mathcal{A}$ or further spend tx and tx$'$ with other users.

Dissolving Fork: Once $\mathcal{A}$ receives the products from both A and B, $\mathcal{A}$ releases the longer chain C_1 or C_2 to all the *reachable* nodes N_r in the Bitcoin network. Complying with the longest chain rule [22, 31], all mining and non-mining nodes switch to the longer chain. $\mathcal{A}$ double-spends since tx$'$ is invalidated.

Figure 15: Generalized construction for SyncAttack. $\mathcal{A}$ orchestrates mining on two blockchain branches and generates conflicting transactions on each branch. When $\mathcal{A}$ receives the reward for each transaction, $\mathcal{A}$ releases the longest branch to diffuse the fork. Note that despite diffusing the fork, $\mathcal{A}$ still controls N_i and can always re-launch the attack.

the consistency property, $\mathcal{A}$ also wastes ≈50% of the network's hashing power.

C DEANONYMIZING TRANSACTIONS

Before 2015, Bitcoin used a gossip-style protocol known as *trickle spreading* to relay transactions among nodes [21]. In *trickle spreading*, a node generates a transaction and relays that transaction to all connections. As such, if an adversary connects to all the *reachable* nodes in the network, the adversary can link a transaction to the *reachable* node's IP address[15]. In 2015, Bitcoin replaced the *trickle spreading* with *diffusion spreading* in order to preserve transaction anonymity [21]. In *diffusion spreading*, the source node waits for a random exponential delay before relaying transactions to each connection. Due to random delay, an adversary may receive a transaction from a different node before receiving the same transaction from the source node. Therefore, *diffusion spreading* increases the transaction anonymity by potentially obfuscating the source node.

However, as noted in [21], an adversary can weaken the anonymity guarantees of *diffusion spreading* by establishing multiple connections to the source node. In [21], the authors implicitly assumed that detecting the source node through such an attack requires an adversary to establish multiple connections through different IP addresses. However, as we have shown in §5.2, the adversary can establish multiple connections using the same IP address, making the attack more feasible. Moreover, using the same node to establish multiple connections also increases the source node detection

[15]Transaction linking in *trickle spreading* is analogous to block linking through which prior works identified the mining nodes [6, 37]

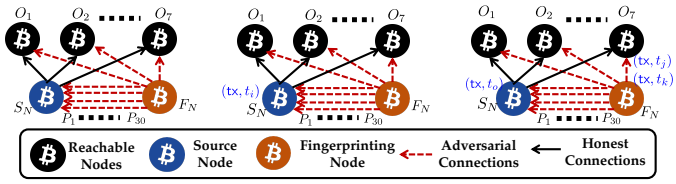

Figure 16: Experiment setup for transaction deanonymization. The source node S_N and the fingerprinting node F_N were connected to the same set of seven nodes ($O = O_1..O_7$). Additionally, F_N established 30 connections to S_N (1 IP and 30 ports). When S_N generated a transaction tx at t_o, we evaluated if any of the 30 connections by F_N received tx from S_N before receiving tx from O.

accuracy since multiple nodes with different IP addresses can experience non-homogeneous propagation delay. In the following, we present our experiments to deanonymize Bitcoin transactions.

Experiment Setup. We set up a source node S_N in the testnet. S_N established outgoing connections to seven *reachable* nodes ($O = O_1, ..., O_7$), and we executed *getpeerinfo* RPC command to obtain their IP addresses. Next, we deployed our fingerprinting node F_N that established one outgoing connection to each node in O, and 30 connections ($P = P1, ..., P30$) to the source node S_N. All 30 connections were established using the same IP address and a different port. From S_N, we generated a series of transactions that were then relayed to O and F_N, following the *diffusion spreading*

protocol. We denoted each transaction as a tuple (tx, t_o), where tx is the transaction identifier and t_o is the transaction generation time. When S_N relayed the transaction with a random delay, the transaction arrived at F_N through (1) one of the 30 connections in P, and (2) one of the seven connections in O. We denoted (tx, t_i) as earliest timestamp of receiving tx from one of the connections in O, and (tx, t_j) as the earliest timestamp of receiving (tx from one of the connections in P. Figure 16 illustrates the experiment setup and the transaction relay procedure. After generating 19 transactions, we applied **Heuristic 3** to deanonymize transactions.

Heuristic 3. *F_N correctly maps transactions to S_N if (1) for all transactions generated by S_N and relayed by S_N and O to F_N, $t_i > t_j$, and (2) for all transactions not generated by S_N but relayed by S_N and O to F_N, $t_i < t_j$.*

The first condition in **Heuristic 3** specifies that if S_N is the transaction source, then F_N receives the transaction from S_N (through one of the 30 connections), before receiving that transaction from O. The second condition in **Heuristic 3** specifies that S_N is not the transaction source, then F_N receives that transaction from O before receiving that transaction from S_N. The second condition is admissible in our experiment settings since we know that S_N is only connected to F_N and O. Therefore, if S_N is not the transaction source, and it relays the transaction to F_N, then clearly S_N received the transaction from O. Based on **Heuristic 3**, the transactions that did not satisfy the first and second conditions were false positives and false negatives, respectively.

Results. We generated 19 transactions from S_N out of which 13 transactions satisfied the first condition in **Heuristic 3** ($t_i > t_j$). Moreover, during the experiment, F_N also received 15 transactions that were not generated by S_N. Among those 15 transactions, 14 transactions satisfied the second condition in **Heuristic 3** ($t_i < t_j$). Overall, among the total 34 transactions, 27 transactions satisfied the two conditions in **Heuristic 3**. Therefore, our experiment of detecting the transaction source achieved an accuracy of ≈79.4%. Per [21], prior works on *trickle spreading* (the weaker anonymity model) achieved an accuracy of ≈30%. In contrast, our experiments on *diffusion spreading* (the stronger anonymity model) achieved an accuracy of ≈79.4%. We suspect that a key contributor to accuracy was using the same node to establish multiple connections, thereby minimizing the non-homogeneous delay.

D REDUCING THE HASH RATE

In addition to double-spending and transaction deanonymization, SyncAttack can also be used to reduce the *effective* hash rate of the Bitcoin network [19]. In §5.1, we outlined three functionalities for $a_i \in A_u$, including the capability of requesting blockchain data from the *reachable* nodes. If $\mathcal{A}$ occupies all the incoming connections of $n_i \in N_i$ and continuously requests the blockchain data from each connection using either the `getblock` or `getheaders` request, then n_i will incur the overhead of processing each request and relaying data to each connection. The problem becomes worse if $n_i \in M_e$ is a mining node, where any unnecessary delay in block relaying reduces the miner's *effective* hash rate [19]. In the following, we investigate how $\mathcal{A}$ reduces the *effective* hash rate of the network.

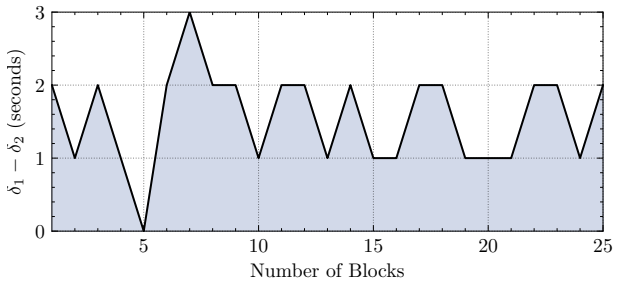

Figure 17: Delay caused by requesting the blockchain data from a victim node through each connection. Our measurements show that on average, a delay of ≈1.56 seconds can be added by overwhelming the bandwidth of the target node.

Experiment Setup. We developed a script with all the functionalities specified in §5.1. We then set up two *reachable* nodes, N_1 and N_2, each maintaining an up-to-date blockchain and establishing 8 outgoing connections to the same set of 8 *reachable* nodes. All other experiment conditions (*i.e.,* bandwidth and processing power) were kept the same for both nodes. We used the *getchaintips* RPC command to record the time at which both nodes received blocks from their outgoing connections. We defined δ_1 and δ_2 as the two timestamps at which N_1 and N_2 received the same block, respectively.

Launching the Attack. Using the lightweight scripts on our local machine, we occupied all the incoming connections of N_1 using 1 IP address and 115 ports. Each connection sent `getheaders` requests to N_1, adding up to 115 parallel requests per second. Upon receiving the block headers from N_1, the connecting node simply discarded those headers and generated new requests. Concurrently, we recorded δ_1 and δ_2 values when N_1 and N_2 received a new block from their outgoing connections.

Results. We conducted the experiment for 26 consecutive blocks. In Figure 17, we plot $\delta_1 - \delta_2$ against the number of blocks. Our results show that N_1 experienced ≈1.56 seconds of additional delay in receiving a block, on average. The maximum and minimum delay values were recorded to be 3 and 0 seconds, respectively. By inspecting the Bitcoin Core source code (*net.cpp* in [11]), we discovered that Bitcoin implements a round-robin request processing for each connection. Therefore, an extra delay of ≈1.56 seconds could have occurred while processing and relaying a response to the bulk of `getheaders` requests sent by the connected node.

Impact on Hash Rate. If $\mathcal{A}$ attacks all the mining nodes, we can expect that each mining node will be delayed by ≈ $\delta_1 - \delta_2$ seconds in receiving blocks from other mining nodes. As a result, the mining node will perform unnecessary computations for additional $\delta_1 - \delta_2$ seconds on a block that is already mined by another node. Keeping in mind that kt is the block propagation delay (3) and $\delta_x = \delta_1 - \delta_2$ is the delay incorporated by $\mathcal{A}$, the *effective* hash rate $\mathcal{H}_e$ of the entire network becomes $\mathcal{H}_e = 1 - \frac{kt+\delta_x}{\lambda}$. Since λ=600 seconds, δ_x decreases $\mathcal{H}_e$ by 0.26% for any value of kt. In other words, in addition to double-spending and transaction deanonymization, $\mathcal{A}$ can reduce the hash rate of the Bitcoin network by 0.26%.

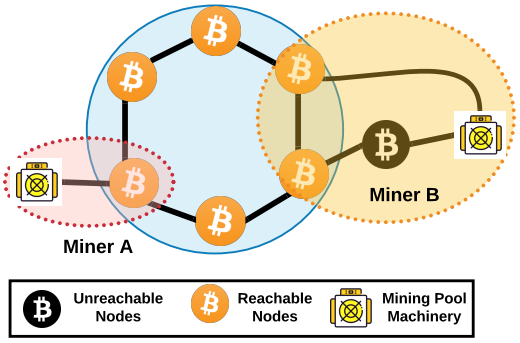

Figure 18: The Bitcoin network anatomy exemplified by the *reachable* and *unreachable* nodes. Miner A owns a *reachable* mining node while Miner B hosts both *reachable* and *unreachable* mining nodes. Blocks between the mining nodes are relayed by the intermediary *reachable* nodes.

Algorithm 2: Determining $\deg^{+}_{\min}(N_r)$

Input: *reachable* nodes N_r
Blockchain $C = (c_1, c_2, ..., c_z)$
Initialize: Object O_{syn} with N_r as keys
foreach $c_j \in C$ **do**
  foreach $n_i \in N_r$ **do**
    if the latest block b_r of $n_i = c_j$ **then**
      append 1 to $O_{\text{syn}}[n_i]$
    else
      append 0 to $O_{\text{syn}}[n_i]$
Return: O_{syn}

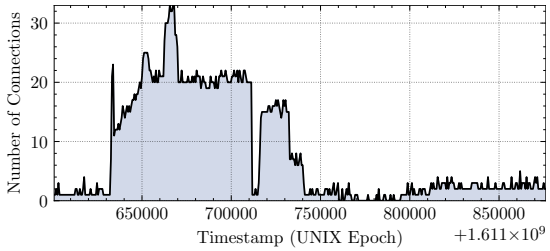

Figure 19: Number of incoming connections from the same IP address recorded on our *reachable* node. We observed instances where the node received up to 33 connections from the same address, indicating an attempt to target our node.

E ADDITIONAL RISKS OF SYNCATTACK

During our measurements in §6.1, we also observed additional risks associated with the weaknesses in the current Bitcoin deployment. In the following, we discuss those risks.

Bitcoin implements a peer *banscore* policy to prevent DoS attacks on the nodes [11]. If a node sends messages that violate the Bitcoin protocol specifications (*i.e.,* invalid signatures), the *banscore* of that node is increased with each message. If the *banscore* reaches 100 the node is disconnected. Theoretically, this means that one IP address can send up to 100 malicious messages to a target node. However, in practice, by exploiting the existing weaknesses, an adversary can send up to $115 \times 99 = 113,85$ such messages while staying connected to the victim.[16]

Moreover, given that the adversary has a complete control over the communication in N_i, the following attacks can be launched to harm users in N_i.

The adversary can selectively relay transactions of high profile nodes (*i.e.,* a Bitcoin exchange) to let only few transactions reach the miners and get mined in the blockchain. The adversary can also stop relaying those transactions altogether to halt trading activities on the exchange and deteriorate the user experience. (1) The adversary can increase delay in the block propagation [23] of a target mining pool to further reduce its *effective* mining power. As a result, the mining pool might be forced to switch to another cryptocurrency. (2) The adversary can violate the chain growth and chain quality properties of the Bitcoin blockchain [34] by reducing the *effective* mining power to bring down the PoW *difficulty* and then increase the *effective* mining power while the *difficulty* remains stable.

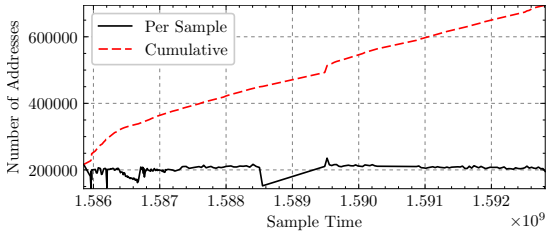

Figure 20: Longitudinal analysis of *unreachable* addresses. The black line shows the unique IP addresses collected in each experiment and the red line shows the cumulative number of unique IP addresses collected in 60 days. The gap between the two lines shows that in each experiment, new IP addresses appeared in the network. Overall, we collected ≈694K unique IP addresses of *unreachable* nodes, with ≈195K nodes present in the network at any time.

F SUPPLEMENTARY FIGURES

In this section, we provide all the supplementary figures that were omitted from the main paper due to space limitations. Each figure is sequentially presented as it appears in the main paper. The figures include (1) Figure 18 used in §2 to illustrate the Bitcoin network anatomy, (2) algorithm 2 used in §4.1.2 to determine if $\deg^{+}(N_r)$ falls below $\deg^{+}_{\min}(N_r)$ for a long period of time, (3) Figure 19 used in subsection 6.1 to show the number of incoming connections from the same IP address received at our node, and (4) Figure 20 used in §5.2.4 to show the number of *unreachable* IP addresses collected to construct **[C2]**.

[16]The default *banscore* is 100. Some nodes can manually increase or decrease the *banscore*. For simplicity, we assume *banscore*=100.