HashSplit: Violating Common Prefix and Chain Quality of the Bitcoin Blockchain

***Abstract*—The safety of Bitcoin blockchain relies on strong network synchrony assumption. Therefore, violating the blockchain safety requires strong adversaries who control an ISP or a mining pool with 51% hash rate. In this paper, we show that the network synchrony does not hold in the real world Bitcoin network, which can be exploited to amortize the cost of various attacks. Towards that, first we construct the Bitcoin ideal world functionality to formally specify its ideal execution model. We then develop a large-scale data collection system through which we connect with more than 36K IP addresses of the Bitcoin nodes and identify 359 mining nodes. We contrast the Bitcoin ideal functionality against real world measurements to expose network anomalies that can be exploited to optimize the existing attack models. Particularly, we observe high block propagation delay in the Bitcoin network causing weak network synchronization: on average, in 9.97 minutes, only 39% nodes have the up-to-date blockchain. Through a fine-grained analysis, we further show non-uniform block propagation delay among the mining nodes to show that the Bitcoin network is asynchronous.**

To realize the threat of asynchronous network, we present the *HashSplit* attack that allows an adversary to orchestrate concurrent mining on multiple branches of the blockchain to violate common prefix and chain quality properties. We also propose the attack countermeasures by releasing a Bitcoin Core version that closely models the Bitcoin ideal functionality. Our measurements, theoretical modeling, and proposed attack and defenses open new directions in the security evaluation of Bitcoin and similar blockchain systems.

I. Introduction and Related Work

Bitcoin is a dynamically evolving distributed system that has significantly scaled up in recent years [12]. As Bitcoin continues to grow and inspire other decentralized applications, its security features are continuously investigated using theoretical analysis and measurement techniques [25], [29]. However, as evident from the prior work, various aspects of theory and measurements have not been combined under a unified framework to fully characterize the Bitcoin network anatomy and synthesize a computation model that captures the intricacies of its real world deployments. We bridge this gap by formally contrasting Bitcoin's theoretical underpinnings with network-wide measurements to investigate its security. To put our work in the appropriate context, below we briefly discuss some notable related works and their limitations.

Theoretical Models' Shortcomings. The existing theoretical models [27], [16], [30], [31] that formally analyze the Nakamoto consensus 1) ignore the mining power centralization in the real world Bitcoin implementation, and 2) implicitly assume a form of synchronous execution that uniformly applies to all network nodes. However, the proof-of-work (PoW) *difficulty* has considerably increased over the years, allowing only a few nodes to mine blocks. As a result, the network is naturally divided between mining and non-mining nodes [4], [26].[1] To incorporate the mining centrality in a theoretical model, we construct the Bitcoin ideal functionality (§II), which acknowledges the distinction between the mining and non-mining nodes and presents an execution model that preserves the blockchain safety properties.

Another limitation of the existing theoretical models is that they assume uniform block propagation delay. The Bitcoin backbone protocol, proposed by Garay *et al.* [16], assumes a *lock-step* synchronous network with no block propagation delay. This assumption is impractical for a large-scale distributed system such as Bitcoin, where block propagation incurs non-zero delay [13]. To address this limitation, Pass *et al.* [30] extended the work in [16] and analyzed Bitcoin in the *non-lock-step* synchronous settings [31]. The *non-lock-step* synchronous model assumes a network in which all miners experience same block propagation delay which gives a uniform advantage to the adversary over all other miners. Our measurements contradict this assumption by showing that miners receive blocks at different times (Figure 7), which cannot be modeled as a uniform advantage. In §A, we analyze the limitations of these models along with refinements to their assumptions. In §V-B, we conduct experiments to show that the real world execution of Nakamoto consensus in Bitcoin is asynchronous. We note that the change in the execution model affects the network synchronization which is pertinent to ensure the two safety properties, namely the common prefix and the chain quality [16]. In §VI, we show that the asynchronous execution relaxes the requirement to violate these properties.

Measurement Studies. In addition to the theoretical models, notable works on the Bitcoin network measurements have focused on (1) analyzing Bitcoin nodes distribution across autonomous systems (ASes) [4], [33], [17], (2) discovering influential nodes in mining pools [26], [4], and (3) measuring information flow and network synchronization [13], [33]. The security evaluations of these studies proposed (1) partitioning attacks through **BGP prefix hijacking of high profile ASes** [4], (2) majority attacks with less than 51% hash rate (≈49% in [13]), and (3) a combination of the two attacks (*i.e.,* spatio-temporal partitioning in [33] and the balance attack in [28]). In the context of measuring the information propagation in the Bitcoin network, the two related studies to our work are by Decker *et al.* [13] and Saad *et al.* [33].

In 2013, Decker *et al.* [13] conducted the first measurement study to analyze the information propagation in the Bitcoin

[1] Bitcoin network consists of full nodes and the lightweight SPV clients. Among the full nodes, there are mining and non-mining nodes. The mining nodes are used by the mining pools to broadcast blocks in the network. In [26], [4], mining nodes are also called the "gateway nodes" of mining pools. The non-mining nodes do not mine blocks and only maintain the blockchain. Our work focuses on the mining and non-mining full nodes.

network. They concluded that the block size is the dominant factor in block propagation delay. In their measurements, they connected to ≈3K IP addresses and observed that ≈90% of the nodes in the network receive the newly published block within 12.6 seconds on average. In contrast, our measurements reveal a network of ≈36K IP addresses, and show that only ≈39% nodes receive the newly published block in 598.2 seconds. The significant discrepancy is due to (1) the significant increase in the network size from 3K to ≈36K reachable IP addresses over time, and (2) the doubling of the block size, from 500KB to 1MB, over the same period of time. More recently, Saad *et al.* [33] used Bitnodes' dataset and reported increasing block propagation delay compared to Decker *et al.*'s findings. They observed instances where a block only reached ≈30% nodes in 600 seconds. They attributed block propagation delay to the increasing netnetwork and suggested that the delay incentivizes malicious miners to create soft forks. In §IV-B, we contrast these measurements with our work, which reflects an up-to-date view of the Bitcoin network.

Limited Attack Strategies. The attacks proposed in these studies have not been frequently observed in the wild due to strong adversarial requirements. First, their threat models directly inherit the assumptions of theoretical frameworks in [16], [30] and ignore the critical distinction between the mining and non-mining nodes (*i.e.,* in [33], [13], [28]). As a result, their models require the adversary to target all the network nodes. Moreover, the inability to distinguish between the mining and non-mining nodes prevents them from analyzing the block propagation among the mining nodes which exposes the asynchronous network. Therefore, these studies have assumed a synchronous network which only allows limited attack strategies [30]. The key challenge lies in getting visibility into the network intricacies to (1) identify the mining nodes, (2) study the block propagation among them, and (3) uncover the actual execution model. With the aid of such fine-grained measurements and their deviation from the ideal functionality, requirements for existing attacks can be significantly lowered, which we demonstrate in this work.

Splitting Mining Power. As mentioned earlier, the effect of block propagation delay on the Bitcoin blockchain has been discussed in theoretical models and measurement studies. Particularly, in the routing attack in [4], the authors show that BGP attacks can reduce the mining power of the Bitcoin network. Similarly, in [28] Natoli *et al.* used the routing attack model to present a tradeoff between the network delay and the adversary's mining power (also simulated by Gervais *et al.* [18]). However, these attacks rely on **disrupting the network communication to create a split between the mining nodes**. Therefore, they implicitly assume a form of routing attack (*i.e.,* BGP hijacking, DNS poisoning, or eclipse attack [28]) as a prerequisite to introduce network delay and split the mining power. In contrast, our measurements show that the block propagation among the mining nodes already exhibits sufficient delay that can be exploited to split the mining power without disrupting the underlying network. We show that by only leveraging the observed block propagation delay among nodes and selective information broadcast, an adversary can violate the safety properties of the Bitcoin blockchain. Therefore, the *HashSplit* attack is more feasible, practical, and stealthy compared with the existing work.

Contributions and Roadmap. Combining our insight from the theoretical analysis and measurements, we present the *HashSplit* attack, which relaxes the requirements to violate the blockchain safety properties. The underpinnings of the *HashSplit* attack are grounded in systematic theoretical analysis and measurements that represent independent contributions in their own right. Along with the attack and its countermeasures, our work exposes the anatomy and characteristics of the Bitcoin network that are summarized below as the key contributions.

1) We construct the Bitcoin ideal world functionality to formally specify the safety properties of the Bitcoin ledger; the common prefix property and the chain quality property [16] (§II). The ideal world functionality faithfully models the expected functionality of a correct Bitcoin implementation across prevalent deployments in real world Bitcoin network.
2) We deploy crawlers in the Bitcoin network and connect with over 36K IP addresses (§III). We develop heuristics to identify the mining nodes and identify 359 IP addresses of the mining nodes using those heuristics (§IV).
3) We measure the block propagation patterns in the Bitcoin network (§V) and show that that the average Bitcoin block time is 9.97 minutes during which only ≈39% nodes receive the latest block, indicating a high propagation delay and weak network synchronization. Moreover, through a fine-grained analysis of the block propagation among the mining nodes we show that execution of the Nakamoto consensus in Bitcoin is asynchronous §V-B.
4) We show the effect of the asynchronous execution by presenting the *HashSplit* attack which allows an adversary to violate the safety properties of the Bitcoin blockchain. We model our attack for an adversary with 26% hash rate and show that the common prefix property and the chain quality properties are violated with a high probability. We also propose attack countermeasures by developing a Bitcoin Core version that closely models the ideal functionality and resists the network asynchrony.

The *HashSplit* attack is a contrast between the Bitcoin ideal world functionality and its real world implementation. Therefore, in keep with the flow, this paper first introduces the ideal world functionality followed by the real world measurements and the attack. Moreover, the paper includes discussions and conclusions in §VIII, and appendices with supplementary findings in §A–§F.

II. The Bitcoin Ideal World Functionality

In this section, we present the Bitcoin ideal world functionality, which we later contrast with real world measurements to present the *HashSplit* attack. The Bitcoin white paper by Nakamoto assumed a network where each node possessed the capability of solving PoW (1 CPU=1 Vote) [27]. However, over time, the PoW difficulty significantly increased, allowing only a few nodes to solve it. This change occurred due to large mining pools that drive implicit forms of centralization [36]. Since there are fewer mining pools than the number of Bitcoin users, therefore, there are fewer mining nodes in the network.

Ideal World Functionality of Bitcoin

Input: Nodes $\mathbb{N}$ including miners M, blockchain $\mathcal{C}$, and trusted party $\mathcal{F}$. The protocol starts at round $r = r_0$ for a length l. Prior to the execution, each $P_i \in M$ reports its hash rate h_i to $\mathcal{F}$, using which $\mathcal{F}$ computes $\mu_i^{'}$, the expected chain quality parameter for each P_i. $\mathcal{F}$ mandates that $h_i < 0.5H, \forall P_i \in M$; otherwise, $\mathcal{F}$ aborts the execution. When a $P_i \in \mathbb{N}$ broadcasts block b_r at time t_0, it reaches all nodes in $\mathbb{N}$ and $\mathcal{F}$ at the next time index t_1. Therefore, $\mathbb{N} \times \mathbb{N}$ is a fully connected graph. The communication model allows each P_i to talk to any node in $\mathbb{N}$ as well as to $\mathcal{F}$ concurrently.

onStart: The block mining starts in which $P_i \in M$ compete.

- Each round r, each $P_i \in M$ computes b_{r+1} with probability $\frac{h_i}{H}$.
- If $P_i \in M$ finds b_{r+1} before it receives b_{r+1} from any other miner, it broadcasts b_{r+1} to $\mathcal{F}$ and $\mathbb{N}$ (no withholding).

onReceive: On receiving a new block b_{r+1}, $P_i \in M$, $P_i \notin M$, and $\mathcal{F}$ follow the following protocol:

- If $\mathcal{F}$ receives a single block b_{r+1} in the round from $P_i \in M$, $\mathcal{F}$ extends the chain $\mathcal{C} \leftarrow b_{r+1}$.
- If $P_i \notin M$ receives a single block b_{r+1} in round r from $P_i \in M$, $P_i \notin M$ extends the chain $\mathcal{C} \leftarrow b_{r+1}$.
- If $P_i \in M$ receives b_{r+1} from another $m_j \in M$ in round, then P_i stops its own computation for b_{r+1}, extends the chain $\mathcal{C} \leftarrow b_{r+1}$, and moves to the next round to compute the next block using b_{r+1} as the parent block.
- If $\mathcal{F}$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), $\mathcal{F}$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1}$ $\mathcal{C}_2 \leftarrow b_{r+1}^{'}$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ have an equal length.
- If $P_i \in M$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), P_i gives time-based precedence to the blocks. *i.e.,* b_{r+1} is received at t_1 and $b_{r+1}^{'}$ is received at t_2, where $t_2 > t_1$, then P_i only accepts b_{r+1} and discards $b_{r+1}^{'}$ by treating it as an orphaned block. P_i extends the chain $\mathcal{C} \leftarrow b_{r+1}$ and moves to the next round to compute the next block using b_{r+1} as the parent block.
- If $P_i \in M$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), at the same time t_1, P_i tosses a coin and selects one of the two blocks to extend the chain.
- If $P_i \notin M$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), $P_i \notin M$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1}$ $\mathcal{C}_2 \leftarrow b_{r+1}^{'}$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ have an equal length.

onTerminate: On input $(r = r_l)$, $\mathcal{F}$ terminates the execution and proceed towards the evaluation of $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

onQuery: In any round, $\mathcal{F}$ can query each $P_i \in \mathbb{N}$ to report $\text{VIEW}_{\mathcal{C}}^{P_i}$. $\mathcal{F}$ then evaluates the $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ for that round.

onValidate: In any round, to validate $\mathcal{Q}_{cp}$, $\mathcal{F}$ queries each $P_i \in \mathbb{N}$ to report $\text{VIEW}_{\mathcal{C}}^{P_i}$. If $\mathcal{F}$ receives a single ledger $\mathcal{C}$ from all $P_i \in \mathbb{N}$, it considers $\mathcal{Q}_{cp}$ to be preserved. If $\mathcal{F}$ receives more than one ledgers (*i.e.,* $\mathcal{C}_1$ and $\mathcal{C}_2$) from one or more $P_i \in \mathbb{N}$, $\mathcal{F}$ prunes k blocks from $\mathcal{C}_1$ chain and verifies if $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ (*i.e.,* two chains share a common prefix). To evaluate $\mathcal{Q}_{cq}$, $\mathcal{F}$ selects the longest chain among $\mathcal{C}_1$ and $\mathcal{C}_2$, and computes the experimental value of μ_i. If $\mu_i - \mu_i^{'} = \epsilon$ (negligible), $\mathcal{F}$ assumes $\mathcal{Q}_{cq}$ is preserved. Otherwise, $\mathcal{Q}_{cq}$ is violated and some $P_i \in M$ has maliciously contributed more blocks than its hash rate.

Figure 1. The Bitcoin ideal world functionality, closely modeled on the practical implementation of Bitcoin as we largely see it. Only mining nodes $P_i \in M$ participate in the block race. The communication model follows the primitive specifications of [27], [16]. We also distinctly characterize the behavior of mining ($P_i \in M$) and non-mining ($P_i \notin M$) nodes when they receive blocks.

Realizing these changes in the network, we formally define the Bitcoin ideal world functionality to characterize the existing Bitcoin operative model, including the distinctive functionality of the mining and non-mining nodes. The formulation of our ideal functionality is inspired by theoretical models proposed in [16], [30], with necessary adjustments to incorporate the mining centrality. To formulate the *safety* and *liveness* of the blockchain, we adopt the formalism from the Bitcoin backbone protocol [16]. In Appendix §A, we explain the model assumptions and the main theorems derived in [16], [30] by presenting the experimental interpretation of their results in the context of our ideal functionality.

First, we define $\mathbb{N}$ as the set of all reachable IP addresses of Bitcoin nodes. We define $\text{VIEW}_{\mathcal{C}}^{P_i}$ as the blockchain view of a single node $P_i \in \mathbb{N}$, where $\mathcal{C}$ is the blockchain ledger. The Bitcoin backbone protocol [16] states that the inter-arrival time between two blocks must be sufficiently large that each $P_i \in \mathbb{N}$ has $\text{VIEW}_{\mathcal{C}}^{P_i}$ (*i.e.,* in ≈10 minutes, all $P_i \in \mathbb{N}$ have the up-to-date blockchain). Next, we define $\{M \subset \mathbb{N}\}$ as a set IP addresses of the mining nodes.[2] For each $P_i \in M$, h_i is P_i's hash power, where $0 < h_i < 1$. $H = \sum_i^{|M|} h_i = 1$ is the total hash power of all the mining nodes. With the network entities defined, below, we discuss the common prefix property and the chain quality property of the Bitcoin blockchain.

Common Prefix Property. The common prefix property $\mathcal{Q}_{cp}$, with parameter k specifies that for any pair of honest nodes P_1 and P_2, adopting the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ at rounds $r_1 \leq r_2$, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$. In this context, an honest node is a node that respects the ideal functionality. $\mathcal{C}_1^{\lceil k}$ denotes the chain obtained by pruning k blocks from $\mathcal{C}$, and $\preceq$ is the prefix relationship. For transaction confirmation, the common prefix property must hold for 6 blocks ($\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ for $k = 6$) [9].

Chain Quality Property. The chain quality property $\mathcal{Q}_{cq}$ with parameters μ and l specifies that for any honest node P_i with chain $\mathcal{C}$, it holds that for any l consecutive blocks of $\mathcal{C}$,

[2] $M = \mathbb{N}$, implies all nodes are the mining nodes (satisfying Nakamoto's assumption). However, we show in §IV, due to centralization, among 36K IP addresses, only 359 belong to mining nodes ($|M|$=359 and $|\mathbb{N}|$=36K)

the ratio of honest blocks is at least μ. $\mathcal{Q}_{cq}$ ensures that for a sufficiently large value of l, the contribution of P_i in $\mathcal{C}$ is proportional to h_i. Moreover, $\mathcal{Q}_{cq}$ assumes that no $P_i \in M$ acquires more than 50% hash rate [15], [23], [19], [21], [34].

Using these properties, we define the Bitcoin ideal world functionality in Figure 1. Our formulation assumes $P_i \in \mathbb{N}$ as "interactive Turing machines" (ITM) that execute the Nakamoto consensus for l rounds, arbitrated by a trusted party $\mathcal{F}$. A round is a time in which each $P_i \in M$ is mining on the same block. For any $P_i \in M$, a round terminates when the $\text{VIEW}_{\mathcal{C}}^{P_i}$ is updated with a new block. The network $\mathbb{N} \times \mathbb{N}$ is a fully connected such that when a block is released by any $P_i \in M$ at t_1, all nodes receive it at the next time step t_2. As a result, the network exhibit a *lock-step* synchronous execution [31]. Due to varying roles in the system, the mining nodes $P_i \in M$ and the non-mining nodes $P_i \notin M$ have unique operations. For instance, when a $P_i \in M$ receives two valid blocks for the same parent block, it gives time-based precedence to the block received earlier. The block received later is discarded. However, when a $P_i \notin M$ receives two valid blocks, it creates two concurrent branches of the chain and waits for the next block to extend one of them. The ideal world functionality in Figure 1 is consistent with the rules encoded in the current Bitcoin Core version. In Appendix §A, we provide the proof for the ideal-world functionality.

Key Takeaways. The ideal world functionality, in Figure 1, characterizes the *modus operandi* of Bitcoin. Compared to the prior theoretical models [16], [30], we distinctly define the mining nodes and non-mining nodes and characterize their unique roles in the system. In the rest of the paper, we perform a data-driven study to investigate 1) the size $|M|$ of the mining nodes, 2) the synchronization patterns in the network to understand how closely Bitcoin follows the ideal functionality, and 3) show deviations to construct the *HashSplit* attack.

III. Data Collection

In this section, we present our data collection system which we used to measure the Bitcoin network. Before describing our data collection system, it is important to discuss the anatomy of the Bitcoin peer-to-peer network.

A. Bitcoin Peer-to-Peer Network

There are two types of Bitcoin full nodes, namely the *reachable* nodes and the *unreachable* nodes. The *reachable* nodes make outgoing connections as well as accept incoming connections. The *unreachable* nodes (often behind a NAT [26]) only make outgoing connections. For simplicity, we can split the Bitcoin peer-to-peer network between the *reachable* space and the *unreachable* space, as shown in Figure 2. Nodes in both the *reachable* and *unreachable* space can only connect with nodes in the *reachable* space.

It is argued that mining pools prefer to host their mining nodes in the *unreachable* space due to security concerns [26]. As such, if we assume that *all* mining nodes exist in the *unreachable* space, that implies mining nodes cannot make direct connections with other mining nodes, and their blocks will have to traverse non-mining nodes in the *reachable* space

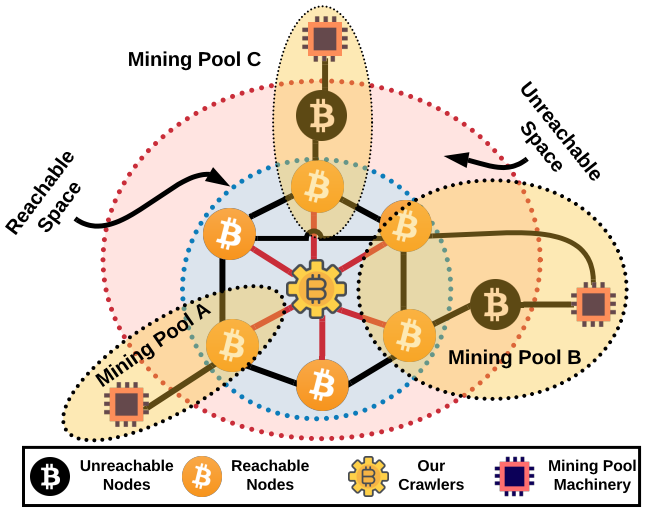


Figure 2. An illustration of our data collection system contextualized in the Bitcoin framework. Mining pools can have reachable (Mining Pool A), unreachable (Mining Pool C), or both (Mining Pool B) types of nodes. Note that unreachable nodes cannot connect with each other. Therefore, a block must appear in the reachable space to reach other miners. Our crawlers directly connect with all the reachable nodes to receive their blocks immediately.

to reach other mining nodes. This assumption alone reflects an asynchronous network that deviates from the ideal world functionality. However, a counter-argument to this assumption states that hosting all mining nodes in the *unreachable* space is undesirable for a mining pool since the pool will ideally want to quickly exchange blocks with other mining pools to avoid selfish mining attacks. To further understand these arguments, we reached out to Bitcoin developers and authors of prior work above. From these discussions, we learned that there is no empirical evidence to support the argument that all the mining nodes exist in the *unreachable* space. In fact, mining pools host both *reachable* and *unreachable* mining nodes. From those discussions, we made the following characterizations.

1) Mining pools host their mining nodes in both *reachable* and *unreachable* space. 2) If a mining pool does not host a mining node in the *reachable* space, it must connect to a non-mining node in the *reachable* space to push its block in the network. 3) A block must traverse the *reachable* space to extend the blockchain of all nodes. 4) If we connect to all the *reachable* nodes in the Bitcoin network, we can observe the entry point of a block. We can treat the entry point as a mining node (*i.e.,* the representative of the mining pool behind it). 5) The frequency of broadcasting a block can be used to estimate the hashing power of the mining pool behind a node.

Using this knowledge, we set up a data collection system to connect with all the *reachable* Bitcoin nodes. Based on the prior work, we noticed that the number of *reachable* nodes in Bitcoin can vary between ≈7K to ≈9K addresses at any time. However, unlike [4], [33], we did not want to rely purely on Bitnodes [10] for data collection since Bitnodes does not disclose its network sampling rate. Instead, we developed our own data collection system from the ground up and customized it to our desired specifications and measurements.

Key Challenges. While setting up our data collection system,

we encountered several challenges. The default Bitcoin Core client is not designed to support large-scale network measurements. The maximum connectivity limit in the Bitcoin Core is 125 (117 incoming and 8 outgoing connections), which is insufficient to map a network of thousands of nodes. Typical stand-alone systems do not support concurrent connectivity with thousands of IP addresses due to file descriptors and socket connection limits. Moreover, to avoid storage intensive network traffic monitoring required for obtaining IP addresses of *reachable* nodes through GETADDR and ADDR messages and for identifying the mining nodes through a block broadcast, we instead leveraged useful artifacts in the Bitcoin Core to maintain a lightweight data collection system. Among them, first we noticed a *peers.dat* file in the Bitcoin Core data directory. The *peers.dat* file compactly logs the information obtained from ADDR messages. The ADDR messages include IP addresses that can be used to expand the network reachability. We developed a *peers.dat* parser to obtain those addresses (details in §C). Second, we used the Bitcoin RPC API for mining nodes detection. In Appendix §B, we provide more details about challenges in our data collection. During this measurement study, we sought help from the Bitcoin Core developers to understand the workings of the software.

B. Data Collection System

We deployed eight crawlers in the Bitcoin network to connect with all the *reachable* nodes. At each crawler, we mounted a NodeJS implementation of the RPC client-server module for data collection and analysis. We also set up a *node manager* and installed the *peers.dat* parser on it. The *node manager* 1) connected to all the crawlers, 2) provided them the list of IP addresses to connect with, 3) obtained the JSON data from each crawler, 4) applied techniques to identify the mining nodes, and 5) measured the block propagation patterns at specified intervals to monitor the network synchronization. In five weeks, we connected to 36,360 unique IP addresses, including 29,477 IPv4, 6,391 IPv6, and 522 Tor addresses. Figure 2 provides an illustration of our data collection system in the context of Bitcoin peer-to-peer network. We connected to all the *reachable* nodes in the Bitcoin network and whenever a mining pool pushed a block in the *reachable* space, our crawlers were able to detect the block and annotate the node.

At each crawler, we used high-speed fiber-optic Internet with a 1GBps connection to minimize propagation delay that could affect measurement results. After five weeks, we discontinued the experiment since we did not observe any increase in the number of mining nodes. Continuing the experiment would not have yielded more meaningful results while continuously surmounting the connectivity cost. At any moment, the crawlers were connected to ≈10K IP addresses, consuming significant bandwidth. Constrained by resources, we limited the experiment duration until we had sufficient data to motivate for the *HashSplit* attack. Complying with the ethical practices, we did not remove all crawlers immediately. Instead, we slowly terminated connections at each crawler to avoid risking any significant effect on the network's health. To avoid influencing block propagation and network synchronization, we disabled block forwarding at our crawlers. Finally, we want to emphasize that our crawlers were merely *listeners* in the network since they only logged the information that was willingly disclosed by their connections. We did not send any measurement probes other than what is acceptable within the Bitcoin network (*i.e.,* GETDATA message in response to a block INV message). When a mining node sent a block through INV or CMPCTBLOCK method (see Figure 12 in §B for details), our crawlers logged the information and took the network snapshot at the time of receiving the block message.

IV. Identifying the Mining Nodes

This section presents the methodology of identifying IP addresses of the mining nodes M. The ratio between $|M|$ and $|\mathbb{N}|$ exposes the network centralization and can be used to relax the adversarial requirements for partitioning attacks [33].

A. Detection Technique and Results

Prior works have used block INV messages to detect mining nodes [4]. This detection method is storage-intensive and Bitcoin Improvement Proposal (BIP152) implementation [11], relying solely on block INV messages can lead to incomplete results. To overcome these limitations, we used the Bitcoin RPC API to sample the network information and developed **Heuristic 1** to detect the mining nodes. We also validated the correctness of **Heuristic 1** using direct network monitoring (see §D). The Bitcoin RPC API command *getblockchaininfo* provides information about the latest block on the blockchain tip. When a new block is received, the chain height is incremented by one, and the new block is added to the chain. We deployed a socket listener at the RPC client-side implementation to record the arrival of a new block from a mining node. When a new block was received, it generated an interrupt on the listener which invoked the *getpeerinfo* API. The *getpeerinfo* renders the up-to-date interactions with all connected peers. In other words, *getpeerinfo* provides a network snapshot as viewed by our crawler. A sample interaction with one peer is shown in Figure 4, and the key variables to note are "addr", "lastsrecv", "synced_headers", and "inflight." "addr" is the connected peer's IP address, "lastrecv" is the latest UNIX timestamp at which the peer relayed any information, "synced_headers" is the peer's blockchain height, and the "inflight" is the block relayed by the peer. Viewed through the lens of the ideal functionality, "synced_headers" renders the view $\textsc{View}_{\mathcal{C}}^{P_i}$ of a peer P_i with the chain tip at $\mathcal{C}$. An update on the tip $\mathcal{C}+1$ is captured by "synced_headers" and "inflight". Using this information, we developed **Heuristic 1** to discover the mining nodes.

Heuristic 1. For a peer P_i, when the blockchain view is updated from $\textsc{View}_{\mathcal{C}}^{P_i}$ to $\textsc{View}_{\mathcal{C}+1}^{P_i}$, if the "synced_headers" value and the inflight value are equal to $\mathcal{C}+1$, then the "addr" value is the mining node $P_i \in M$'s IP address.

Heuristic 1 is a mapping between the information exposed by the RPC API and the Bitcoin network traffic of a crawler. For more clarity on **Heuristic 1**, revisit Figure 4 that shows a sample interaction of a crawler connected to a set of peers in $\mathbb{N}$ with its blockchain tip $\mathcal{C}$ = 570366 ("synced_blocks"=570366).

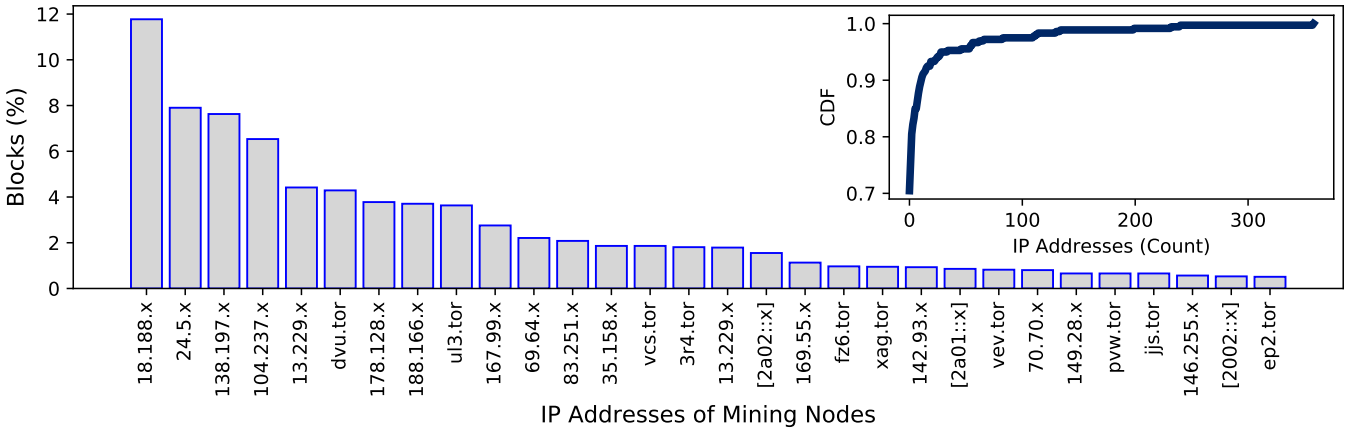


Figure 3. Results obtained after applying Heuristic 1 on our dataset. The histograms show the percentage of blocks contributed by top IP addresses. We mask the last two octets to preserve anonymity. The subplot is the CDF showing the distribution of IP addresses with respect to the total blocks produced.

```
{ id: 12188,
  addr: '169.x.x.x:8333',
  addrlocal: '132.x.x.x:8333',
  addrbind: '132.x.x.x:8333',
  lastsend: 1554493200,
  lastrecv: 1554493185,
  conntime: 1554002430,
  timeoffset: 31,
  pingtime: 0.118272,
  minping: 0.05748,
  version: 70015,
  subver: ''/Satoshi:0.16.0/'',
  startingheight: 569534,
  synced_headers: 570367,
  synced_blocks: 570366,
  inflight: [ 570367 ], }
```

Figure 4. A truncated sample JSON output when a block is received by our crawler from a network peer. Here, "addr" is the IP address of the peer to which the crawler is connected to, "synced_headers" is the height of blockchain header at which the crawler has synchronized with the node, and "inflight" is the block that the node is transmitting to the crawler.

When the crawler receives an update "570367" from *getblockchaininfo*, it checks $\text{VIEW}_{\mathcal{C}}^{P_i}$ of all its connected peers using *getpeerinfo*. One information sample of a connected peer is shown in Figure 4. For each peer, the crawler checks if "synced_headers" value is 570367 ($\mathcal{C}+1$). When the mining node relays a block, the "inflight" value is also set to the block height ($\mathcal{C}+1$). The example in Figure 4 shows that the "inflight" value is "570367", hence the "addr" value is the mining node's IP address.

Results. We applied **Heuristic 1** to detect the mining nodes. A summary of results is reported in Figure 3. To preserve anonymity of the mining nodes, we mask the last two octets of their IP addresses. Overall, we discovered 359 mining nodes. Among them, 250 (69.6)% were IPv4, 34 (9.47%) were IPv6, and 75 (20.89)% were Tor addresses. Our results indicate that mining pools use Tor to shield their nodes from routing attacks [3]. Among the 359 mining nodes, 31 nodes produced 80% blocks, and 67 nodes produced 90% blocks.

B. Revisiting Partitioning Attacks

We revisit the two notable partitioning attacks in the literature [33], [4] to understand how the new insight we uncovered about the mining nodes and their characterization affect those attacks threat model assumptions.

Temporal Partitioning Attacks [33]. In the temporal partitioning attack, an adversary connects to all reachable IP addresses in $\mathbb{N}$ to isolate the vulnerable nodes that have an outdated view. As such, the threat model in [33] makes no distinction between the mining and non-mining nodes. Therefore, attacking a mining node becomes a probability game in which the adversary expects that a vulnerable node is one of the mining nodes. However, with additional insights from our work, if the adversary learns about the mining nodes M, the attack can be significantly optimized. Our results show that among $|\mathbb{N}|$=36,360 IP addresses, only $|M|$=359 IP addresses belong to the mining nodes. Since, $|M| << |\mathbb{N}|$, the connectivity overhead can be significantly reduced.

Routing Attacks [4]. Although our results can be used to optimize the temporal partitioning attacks, we observe rather unfavorable outcomes for routing attacks. BGP-based routing attacks are launched to disrupt communication between subgroups of a blockchain system [28]. These attacks are more effective when launched against the mining nodes to split the network hash rate. To analyze this aspect, we monitored the hosting patterns of the mining nodes and made the following observations. Since 20.89% of all the mining nodes use Tor, they are less vulnerable to the BGP hijacks. Among the remaining 284 (79.1%) nodes that use IPv4 and IPv6, 90 (31.69%) mining nodes are uniquely hosted across 90 ASes (one node per AS), showing a high diversity in hosting patterns. We further observed that 15 (5.28%) ASes hosted 2 nodes, and 11 (3.87)% ASes hosted 3-8 nodes. Only 5 (1.76%) ASes hosted 10 or more mining nodes. In Figure 5, we plot the CDF of the mining nodes across ASes. Due to the high diversity in the hosting patterns, the mining nodes are less vulnerable to the routing attacks, as previously reported in [3], [33]. However, these results are not particularly surprising: the

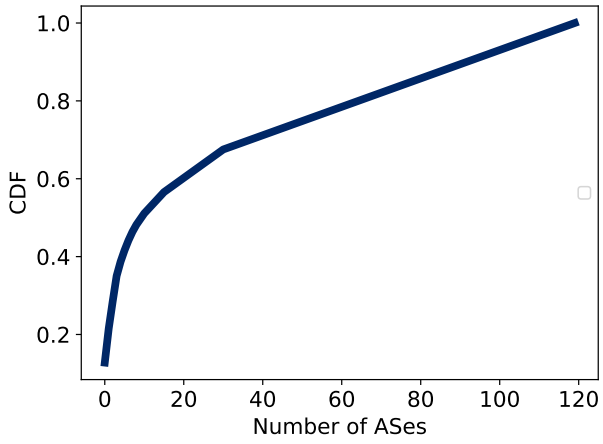


Figure 5. Distribution of mining nodes across ASes. Compared to the general distribution of Bitcoin full nodes [33], [4], mining nodes are comparatively more decentralized, and therefore, less vulnerable to routing attacks. 90 (31.69%) mining nodes are uniquely hosted across 90 ASes (at the time of this study), showing a higher distribution in the hosting patterns.

routing attacks are well-known to the mining pool operators, perhaps leading them to diversify their mining nodes' hosting patterns to protect them against such attacks.

V. Network Synchronization

In this section, we monitor the blockchain synchronization patterns to check if the network complies with the ideal world specifications. To preserve the common prefix property, the inter-arrival time between two blocks must be long enough to allow each node to have an up-to-date blockchain [16]. In other words, when a new block is released, all nodes must have the previous block on their blockchain tip. If a node does not exhibit this behavior (*i.e.,* due to propagation delay), it is vulnerable to partitioning. To concretely evaluate the network synchronization, we use **Heuristic 2** below.

Heuristic 2. When a crawler receives a new block b_{i+1} from a mining node, the crawler checks $\text{VIEW}_{\mathcal{C}}^{P_i}$ for all connected peers in $\mathbb{N}$. For a connected node P_i, if the blockchain tip $\mathcal{C} = b_i$ (the previous block), then P_i is synchronized with an up-to-date blockchain view. If $\mathcal{C} < b_i$, then the peer is behind the chain by 1 or more blocks, showing poor synchronization.

To elaborate **Heuristic 2**, we again refer to Figure 4. When our crawler received the new block 570367 (b_{i+1}) from the mining node, the "synced_blocks" value was 570366 (b_i). This means that before sending the new block, the mining node's blockchain tip was 570366 ($\mathcal{C} = b_{i-1}$). Hence, the mining node had an up-to-date blockchain. Similarly, at the time of receiving the new block 570367, the crawler expected all its connected peers to have the "synced_blocks" value 570366 in order to satisfy the synchronization property [16]. If the value of "synced_blocks" is less than 570366 (*i.e.,* 570365), the peer is behind the blockchain by one block. The "synced_blocks" value of all connected peers was obtained from the RPC API. We sampled this information every time we received a block from a mining node. As a result, we were able to measure the fraction of nodes with an up-to-date blockchain.

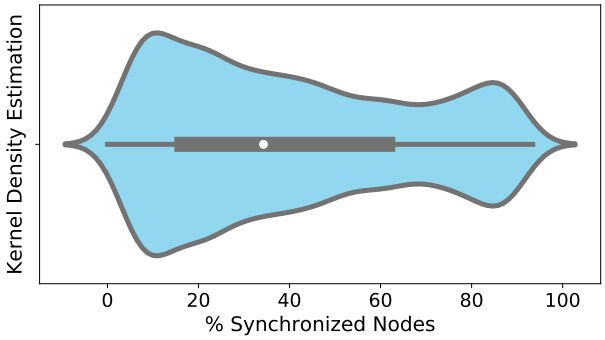


Figure 6. Block synchronization in the Bitcoin network shown as a kernel density plot. On average, by the time a new block is produced, only 39.43% nodes receive the previous block. Variations in the kernel density shape show a non-uniform synchronization pattern due to varying propagation delay.

We report results in Figure 6 as a violin plot [7]. The x-axis shows the percentage of synchronized nodes, and the y-axis is the kernel density estimation rendering the data distribution shape. The wider sections on y-axis show a higher density of data points clustered in the region. To meet the requirements of the ideal functionality, the shape of Figure 6, as well as the mean value, must be close to 100%. On the contrary, our results were considerably different from expectations. On average, only 39.43% nodes had the up-to-date blockchain when a new block arrived, showing a poor network synchronization. During our measurements, the mean inter-arrival time between two blocks was 9.97 minutes (598.2 seconds). In other words, our results showed that, on average, during 9.97 minutes, only 39.43% of all *reachable* nodes received the previous block and had an up-to-date blockchain. The minimum and maximum values for network synchronization were 0.04% and 93.41%, respectively, with a standard deviation of 26.88%.

A. Revisiting Information Propagation

Our results reveal weak network synchronization in the Bitcoin network. Particularly, our measurements are significantly different from prior studies conducted on block propagation in Bitcoin [13], [33]. In the following, we contrast our findings with prior works and explore factors behind increasing delay.

Decker *et al.* [13]. In 2013, Decker *et al.* reported that 90% Bitcoin nodes receive a new block in 12.6 seconds. They also observed a high correlation between the block size and the block propagation delay, stating that for blocks greater than 20KB, each KB increase in the size adds 80 milliseconds delay in the block propagation. In their measurement duration, they connected to ≈3K IP addresses. In our measurement duration, we connected to more than 36K IP addresses, and at any time, our crawlers were simultaneously connected to ≈10K addresses. In 2013, the average Bitcoin block size was 500KB. In our experiments, we observed that the average block size was 1.02MB, with the maximum block size of 1.29MB. In terms of the size-delay relationship established in [13], there is a natural delay of at least 41.6 seconds ($(1020 - 500) \times 0.08 = 41.6$) due to the increase in average

block size. Additionally, since the network has scaled up in the orders of magnitude, there are additional transmission and propagation delays that contribute to weak synchronization.

Saad *et al.* [33]. In 2018, Saad *et al.* [33] used the Bitnodes [10] dataset to observe the synchronization patterns in Bitcoin. In their work, they observed ≈13K Bitcoin addresses through Bitnodes and reported higher block propagation delay than [13]. They also noticed a few cases where blocks only reached ≈30% nodes in 10 minutes. However, since they relied on Bitnodes dataset, therefore they did not have direct visibility into the network and block propagation patterns. Their measurements were constrained by Bitnodes network sampling rate which is not disclosed by Bitnodes. In contrast, we conducted direct network measurements and sampled the network state precisely upon the arrival of a new block. Nevertheless, their study also showed increasing block propagation delay since 2014 [13], which is consistent with our findings.

B. Asynchrony in Bitcoin

High block propagation delay shows that the Bitcoin network significantly deviates from the ideal world functionality. Since the overall network suffers from a weak synchronization, it is therefore intuitive to assume that the mining nodes may also experience a high latency in the block reception as well. Moreover, non-uniform delay in the block propagation indicates an asynchronous network. To the best of our knowledge, the notable work that closely captures this behavior is conducted by Pass *et al.* [30] in which they performed a theoretical analysis to understand the performance of Bitcoin in the *non-lock-step* synchronous execution. This execution model allows an adversary to delay the block by a parameter Δ, giving the adversary a head start mining advantage. Pass *et al.* [30] assumed that after Δ, all the mining nodes simultaneously receive a block to start the next round. More precisely, they assumed that all honest miners *"freeze"* and do not start mining until all miners receive the block. However, in practice, when miners (honest or otherwise) receive a block, they immediately start mining the next block [27]. Therefore, if miners receive blocks at different times and start mining immediately, the network becomes asynchronous rather than *non-lock-step* synchronous. This hypothesis can only be validated by experimentally observing the block propagation among mining nodes, which we do in the following section.

Block Propagation Among Mining Nodes. We performed a follow up experiment to study the block propagation delay among the mining nodes and validate the *asynchronous* execution of Nakamoto consensus in Bitcoin. For that experiment, we took the network snapshot at one second interval. The difference between the two experiments is that in the previous experiment we only took the network snapshot when we observed a new block in the network. In the second experiment, however, we sampled the network every second. All our crawlers executed the *getpeerinfo* command every one second to obtain the $\textsc{View}_{\mathcal{C}}^{P_i}$ of each connected peer. Hence, we were able to record the time when a new block appeared in the network and the time at which each node in the *reachable* space received it (with one second granularity). Since we had the IP addresses of the mining nodes, we were able to note the time at which each mining node received the block.

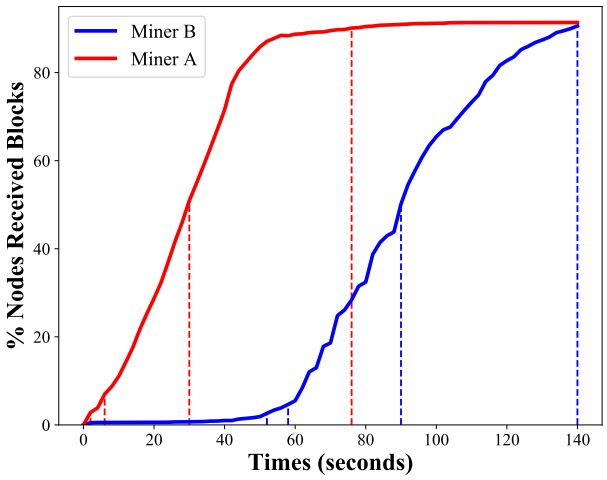


Figure 7. Block propagation of two miners, sampled at one second interval. Note that Miner A's block reaches $|M|/2$, $|M|$, $|\mathbb{N}|/2$, and $|\mathbb{N}|$ at 2, 6, 30, and 76 seconds, respectively. In contrast, Miner B's block reaches the same set of nodes at 52, 58, 90, and 140 seconds respectively. Miner A has a significant advantage over Miner B in terms of block propagation.

In our second experiment, we made three key observations. First, we noticed that the mining nodes received blocks faster than the non-mining nodes. Second, we also noticed that the mining nodes received blocks at different time intervals, confirming asynchronous execution. On average, 25% mining nodes received blocks within 47.45 seconds, 50% mining nodes within 55.21 seconds, and 100% within 62.30 seconds. Third, we observed that the block propagation delay for each mining node varied, showing a variation in the network reachability (*i.e.,* the number of connected peers). Variations in block propagation delay confirms that the mining nodes do not form a completely connected $M \times M$ topology. To further highlight this aspect, we present an illustrative example, in Figure 7 showing block propagation for two mining nodes randomly selected from the second experiment. For simplicity, we label the two mining as "Miner A" and "Miner B." Figure 7 shows that when "Miner A" released the block, within 2 seconds, the block reached half of the mining nodes, and within 6 seconds, the block reached all the mining nodes. Moreover, within 76 seconds, the block reached ≈90% of all the nodes in the network. In contrast, when 'Miner B" released the block, the block took 52 seconds to reach half of the mining nodes and 58 seconds to reach all the mining nodes. The block took 140 seconds to reach ≈90% network. Although, it might appear from this experiment that almost the entire network received blocks within 76 seconds (Miner A) and 140 seconds (Miner B), however, these are just two representative examples presented to highlight the variation in the network reachability. On average, the block synchronization pattern was similar to the results reported in Figure 6. From Figure 7, we derived the following conclusions. (1) Since mining nodes receive blocks at different times, therefore, the execution of the Nakamoto protocol in Bitcoin is asynchronous instead of *lock-step* synchronous [16] or *non-lock-step* synchronous [30], [31].

(2) Logically, this behavior suggests that the mining nodes don not for a completely connected $M \times M$ topology. This is analogous to our illustration in Figure 2, where Mining Pool A is two hops away from Mining Pool B, and Mining Pool C is one hop away from Mining Pool B. If Mining Pool B broadcasts a block through its mining node, Mining Pool C is likely to receive it before Mining Pool A. (3) Variations in the block propagation delay also suggest that the Bitcoin mining nodes have a varying network reachability.

VI. The *HashSplit* Attack

To contextualize the security implications of high block propagation delay and asynchronous network, we present the *HashSplit* attack that allows an adversary to orchestrate concurrent mining on multiple branches of the public chain and violate the common prefix ($\mathcal{Q}_{cp}$) and the chain quality ($\mathcal{Q}_{cq}$) properties. *HashSplit* is a lower bound construction that shows: (1) an adversary with arbitrary hashing power can violate $\mathcal{Q}_{cq}$, (2) an adversary with 26% hash rate can violate both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ with a high probability, and (3) the requirement for a majority attack under any hash rate can be amortized for all computationally admissible Bitcoin executions. From our measurements (§III–§V-B), we note that the asynchronous network creates a natural partitioning among the mining nodes, which when combined with the Bitcoin mining policies [27], [30], [15], [13], [16], expand the strategy space for an adversary to launch various attacks. *HashSplit* is one such attack that targets the blockchain safety properties.

It is worth noting that, unlike the balance attack [28] or the routing attacks [4], [33], *HashSplit* does not require the adversary to disrupt the network communication through BGP hijacking. Instead, the adversary simply exploits the existing propagation delay among the mining nodes and the natural partitioning created by asynchronous network to split the network hash rate. Below, we present the threat model for the *HashSplit* attack. For more details on the attack novelty compared to other Bitcoin attacks, we refer to Appendix §E.

A. Threat Model and Attack Objectives

For *HashSplit*, we assume an adversary $\mathcal{A}_m \in M$ with less than 51% hash rate. $\mathcal{A}_m$ follows the experiment methodology in §III–§V to connect to all $P_i \in \mathbb{N}$, identify the mining nodes M, estimate their hashing power using the block mining rate (Figure 3), and obtain the block propagation pattern of each mining node (Figure 7). After identifying all $P_i \in M$, $\mathcal{A}_m$ maintains a direct connection with them to instantly send or receive blocks. Using the measurement methodology in Figure 7, $\mathcal{A}_m$ calculates how a block generated by each $P_i \in M$ reaches all $P_i \in \mathbb{N}$. If $\mathcal{A}_m$ samples the block propagation pattern of each $P_i \in M$, Figure 7 can be generalized as an illustrative example in Figure 8.

Figure 8 shows that $\mathcal{A}_m$ has good network reachability like Miner A in Figure 7, while $\mathcal{H}_m$ (an honest miner) has poor network reachability like Miner B in Figure 7. Since Figure 7 is sampled at one second interval, therefore $\mathcal{A}_m$ knows precisely (in seconds) at what time each $P_i \in \mathbb{N}$ receives a block. By calculating the difference in the block

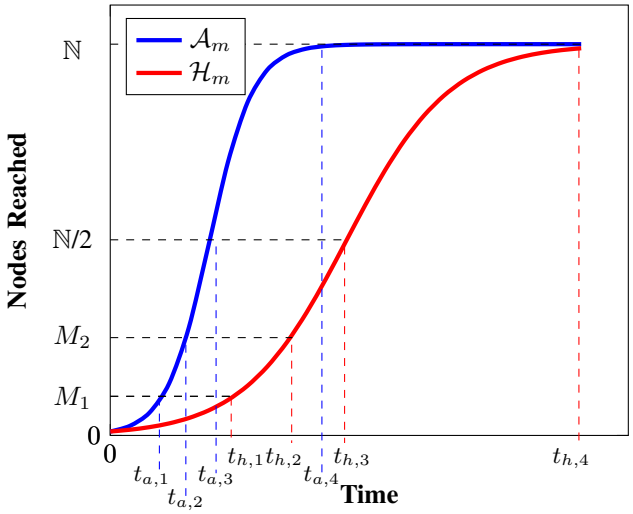


Figure 8. Generalized illustration of Figure 7. For simplicity of modelling, we assume a uniform hashing power distribution in M (*i.e.,* $M_1 \approx |M|/2 \approx 51\%$ hash rate and *i.e.,* $M_2 \approx |M|/2 \approx 49\%$ hash rate). $\mathcal{A}_m$ is well connected compared to $\mathcal{H}_m$. If $\mathcal{H}_m$ and $\mathcal{A}_m$ concurrently produce a block, $\mathcal{A}_m$ wins race due to $\mathcal{H}_m$'s propagation delay. Here $t_{a,1}, t_{a,2}, t_{a,3}$ and $t_{a,4}$ are times when $\mathcal{A}_m$'s block reaches 50% miners, 100% miners, 50% network, and 100% network. Accordingly, $t_{h,1} \ldots t_{h,4}$ are the corresponding values for $\mathcal{H}_m$.

generation time and the time at which each $P_i \in \mathbb{N}$ receives the block, $\mathcal{A}_m$ can calculate the delay in the block reception for each $P_i \in \mathbb{N}$. For simplicity, for each $P_i \in M$, we define the *reachability time* $T_{i,j} = [t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}]$ as four discreet time indexes at which the block is received by 50% miners, 100% miners, 50% network, 100% network.

We further assume that each $P_i \in M$, except $\mathcal{A}_m$, conforms to the ideal functionality such that when any $P_i \in M$ generates a block, it immediately releases the block to the network without withholding. Moreover, when a $P_i \in M$ receives two blocks with a hash pointer to the same parent block, $P_i \in M$ gives time-based precedence to the block received earlier, and mines on top of it. The time-based precedence is a mining policy proposed by Nakamoto [27] and currently deployed in all Bitcoin Core versions. Finally, we assume that (1) $\mathcal{A}_m$ cannot influence the communication model of other $P_i \in \mathbb{N}$ through a routing attack [4], [28], and (2) $\mathcal{A}_m$ there is no other ongoing attack (*i.e.,* selfish mining) parallel to *HashSplit*. We specifically model *HashSplit* for a weaker adversary to present a lower bound construction. Logically, the attack is more favorable for a stronger adversary considered in prior works on Bitcoin partitioning attacks [4], [13], [33], [30].

Attack Objectives. Given that $\mathcal{A}_m$ is a miner with view of network communication model, $\mathcal{A}_m$ can: (1) deviate from the ideal functionality and violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ of the Bitcoin blockchain, (2) waste the mining power of honest miners, (3) prevent non-mining nodes from generating or receiving k-confirmed transactions [27]. In *HashSplit*, $\mathcal{A}_m$ achieves these objectives by exploiting block propagation delay to split the public blockchain chain into two branches $\mathcal{C}_1$ and $\mathcal{C}_2$, and the mining nodes M into two groups M_1 and M_2. In a perfect split, $\mathcal{A}_m$ splits the network hash rate into $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$ ($\alpha + \beta = 1$), and mines on the branch with a

Algorithm 1: Identifying Vulnerable Mining Nodes

Input: $T_{i,j}, T_{a,j}$ (reachability time of the adversary and all other mining nodes)
Initialize: aList, bList, cList, dList
Initialize: aMax, bMax, cMax, dMax = 0
for $i = 0$; $i < |M|$; i++ **do**
  $\delta_1 = t_{i,1} - t_{a,1}$, aList $\leftarrow \delta_1$
  if $\delta_1 >$ aMax **then**
    aMax = δ_1
  $\delta_2 = t_{i,2} - t_{a,2}$, bList $\leftarrow \delta_2$
  if $\delta_2 >$ bMax **then**
    bMax = δ_2
  $\delta_3 = t_{i,3} - t_{a,3}$, cList $\leftarrow \delta_3$
  if $\delta_3 >$ cMax **then**
    cMax = δ_3
  $\delta_4 = t_{i,4} - t_{a,4}$, dList $\leftarrow \delta_4$
  if $\delta_4 >$ dMax **then**
    dMax = δ_4
return: aList, bList, cList, dList, aMax, bMax, cMax

Algorithm 2: Attack Procedure (Split Ledger)

Input: M, $\mathcal{A}_m$
Case 1: $\mathcal{A}_m$ finds b_{r+1} before any $P_i \in M$
Strategy 1: $\mathcal{A}_m$ waits for another $P_i \in M$ to find $b^{'}_{r+1}$. When $\mathcal{A}_m$ receives $b^{'}_{r+1}$ from $P_i \in M$, $\mathcal{A}_m$ releases b_{r+1} only to M_1 before M_1 receive $b^{'}_{r+1}$. $\mathcal{A}_m$ does not release b_{r+1} to M_2, which invariably receive $b^{'}_{r+1}$ from the other miner at $t_{i,2}$ (Figure 8).
Case 2: Any $P_i \in M$ finds $b^{'}_{r+1}$ before $\mathcal{A}_m$
Strategy 2: $\mathcal{A}_m$ violates the ideal functionality (see **onStart** in Figure 1) and keeps mining b_{r+1}. By $t_{i,1}$, $b^{'}_{r+1}$ reaches M_1 miners. If $\mathcal{A}_m$ finds b_{r+1} before $t_{i,1}$, it releases b_{r+1} to M_2 before $b^{'}_{r+1}$ reaches them.
Result: In **Strategy 1**, M_1 receives b_{r+1} and M_2 receives $b^{'}_{r+1}$. In **Strategy 2**, M_1 receives $b^{'}_{r+1}$ and M_2 receives b_{r+1}. In both cases, the chain $\mathcal{C}$ splits into two branches $\mathcal{C}_1$ and $\mathcal{C}_2$, and the network hash rate into α and β. A fork is created.

higher hash rate. To violate $\mathcal{Q}_{cp}$ for any $P_i \in \mathbb{N}$, $\mathcal{A}_m$ ensures that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ for $k = 6$. To violate $\mathcal{Q}_{cq}$, $\mathcal{A}_m$ ensures that for any $P_i \in \mathbb{N}$, $\mu_i - \mu_i^{'} \neq \epsilon$ (blockchain has disproportionately high adversarial blocks). We show that a *HashSplit* adversary meets these objectives with a high probability.

B. Attack Procedure

1) Identifying Vulnerable Nodes: To split the blockchain, $\mathcal{A}_m$ first identifies the vulnerable mining nodes that have a high block propagation delay. $\mathcal{A}_m$ runs algorithm 1 to identify those vulnerable nodes. In algorithm 1, $T_{a,j}$ and $T_{i,j}$ are *reachability times* for $\mathcal{A}_m$ and other $P_i \in M$, respectively. $\mathcal{A}_m$ initializes four lists (aList ... dList) and four variables (aMax ... dMax). For each $P_i \in M$, $\mathcal{A}_m$ computes the time windows $\delta_1 ... \delta_4$ that represent the difference between the block propagation time of the $\mathcal{A}_m$ and the target mining node. For an intuitive understanding, we again refer to Figure 7. Assuming Miner A as $\mathcal{A}_m$ and Miner B as $\mathcal{H}_m$, algorithm 1 outputs $\delta_1 = 50$, $\delta_2 = 52$, $\delta_3 = 60$, and $\delta_4 = 64$ seconds, respectively. Therefore, algorithm 1 provides the difference in the *reachability time* of all $P_i \in M$ relative to $\mathcal{A}_m$'s *reachability time*. Additionally, algorithm 1 also determines the most vulnerable node with the maximum *reachability time* difference, which can be the easiest target to initiate the split.

2) Blockchain Splitting: After discovering the vulnerable nodes, $\mathcal{A}_m$ splits the blockchain into two branches $\mathcal{C}_1$ and $\mathcal{C}_2$, and miners into two groups M_1 and M_2 using algorithm 2. We define the combined hash rate of M_1 as α and M_2 as β. algorithm 2, provides two attack strategies to achieve the split.

Strategy 1. In this strategy $\mathcal{A}_m$ produces a block b_{r+1} before any $P_i \in M$, and withholds it. $\mathcal{A}_m$ waits for another $P_i \in M$ to produce a block $b^{'}_{r+1}$. With apriori knowledge of $b^{'}_{r+1}$ propagation pattern in the network (algorithm 1), $\mathcal{A}_m$, releases b_{r+1} to M_1 while $b^{'}_{r+1}$ reaches M_2. As a result, when $b^{'}_{r+1}$ reaches M_1 after $t_{a,1}$, M_1 will not mine on it (time-based precedence [27]). However, by $t_{i,2}$, M_2 receive $b^{'}_{r+1}$ and start mining on it. Since the miners mine on the earliest received block (b_{r+1} for M_1 and $b^{'}_{r+1}$ for M_2), the blockchain forks into two branches $\mathcal{C}_1 \leftarrow \alpha$ and $\mathcal{C}_2 \leftarrow \beta$.

Strategy 2. In this strategy, an honest miner $P_i \in M$ produces the block $b^{'}_{r+1}$ before $\mathcal{A}_m$. Since $\mathcal{A}_m$ knows that $b^{'}_{r+1}$ will take $t_{i,1}$ time to reach M_1 (see Figure 8), $\mathcal{A}_m$ violates the ideal functionality and keeps mining for b_{r+1} until $t_{i,1}$. If by $t_{i,1}$, $\mathcal{A}_m$ succeeds in mining b_{r+1}, $\mathcal{A}_m$ releases b_{r+1} to the other set of miners (M_2) to which $b^{'}_{r+1}$ is yet to reach. As a result, similar to **Strategy 1**, the blockchain splits into $\mathcal{C}_1 \leftarrow \alpha$ and $\mathcal{C}_2 \leftarrow \beta$. In summary, algorithm 2 provides two strategies for the adversary to split the chain into two branches.

Perfect Split. As described in §VI-A, the perfect split leads to $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$. If $\mathcal{A}_m$ with hash rate α_1 mines on $\mathcal{C}_1$, we define the combined hash rate of all miners in M_1 as $\alpha = \alpha_1 + \alpha_2$. Since $\mathcal{A}_m$ knows the block propagation pattern and the hash rate distribution (Figure 8), $\mathcal{A}_m$ can time both strategies in algorithm 2 to achieve the perfect split such that $\alpha = \alpha_1 + \alpha_2 = 0.51$. For simplicity, and without losing generality, in the rest of the analysis, we assume: (1) $\mathcal{A}_m$ achieves perfect split from algorithm 2, (2) there are four miners in the network namely $\mathcal{A}_m$, h_1, h_2, and h_3, (3) $\mathcal{A}_m$ and h_1 mine on $\mathcal{C}_1$ with $\alpha_1 = 0.26$ and $\alpha_2 = 0.25$, (4), h_2 and h_3 mine on $\mathcal{C}_2$ with hash rates $\beta_1 = 0.25$ and $\beta_2 = 0.24$, respectively ($\beta = \beta_1 + \beta_2 = 0.49$), and (5) $\mathcal{A}_m$ has block propagation pattern similar to Miner A in Figure 7 and all other miners have block propagation patterns of Miner B in Figure 7. Therefore, at $t_{a,1}$, $\mathcal{A}_m$'s block reaches h_1 and at $t_{a,2}$, the block reaches both h_2 and h_3. Similarly, for h_2, $t_{h,1}$ and $t_{h,2}$ are times at which $\mathcal{A}_m$ and both h_2 and h_3 receive a block. We can extend the same propagation sequence for h_2 and h_3. We make these assumptions only to simplify our attack analysis. The model can be easily generalized to more than four miners with varying hash rates and reachability times. Considering that the mining nodes in the actual network often experience more delay than the case in Figure 7, therefore, our lower bound construction holds in practice.

3) Block Race: Once the perfect split is achieved, the two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ enter in a block race. To formally analyze the race conditions, first we revisit the mathematical underpinnings of Nakamoto consensus in Bitcoin. Bitcoin mining can be modeled as a Poisson process with inter-block times exponentially distributed with mean $\tau = 600$ seconds. A valid block has the double hash of the block header less than the difficulty SHA256(SHA256((Header))$< d =\in [0, 2^{256} - 1]$. On average, a miner computes $m = 2^{256}/d$ hashes to mine a block [20]. With the total network hash rate $\alpha + \beta$, $m = (\alpha + \beta) \times \tau$ is the total number of hashes required to mine a block at the specified block time τ [20]. When hash rate splits into α and β (algorithm 2), the time required to mine the next block on each branch becomes $t_o = m/(\alpha)$ and $t_o^{'} = m/\beta$.

Block Race

Fork Persists:

- $\mathbf{f_1}$: $\mathcal{A}_m$ produces a block on $\mathcal{C}_1$. No other miner produces a block on either $\mathcal{C}_1$ or $\mathcal{C}_2$. $\mathcal{A}_m$ withholds its block to maintain the fork. Event probability is $\alpha_1(1-\alpha_2)(1-\beta_1)(1-\beta_2)$.
- $\mathbf{f_2}$: $\mathcal{A}_m$ produces a block on $\mathcal{C}_1$ and either h_2 or h_3 produce a block on $\mathcal{C}_2$. $\mathcal{A}_m$ sends its block to h_1 who mines on $\mathcal{C}_1$. h_1 and h_2 mine on $\mathcal{C}_2$. Event probability is $\alpha_1\beta_1(1-\alpha_2)(1-\beta_2) + \alpha_1\beta_2(1-\alpha_2)(1-\beta_1)$.
- $\mathbf{f_3}$: $\mathcal{A}_m$ produces a block on $\mathcal{C}_1$ and both h_2 and h_3 produce a block on $\mathcal{C}_2$. Three chains appear $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. $\mathcal{A}_m$ sends its block to h_1 and both mine on $\mathcal{C}_1$. h_2 and h_3 mine on $\mathcal{C}_2$ and C_3, respectively. Event probability is $\alpha_1\beta_1\beta_2(1-\alpha_2)$.
- $\mathbf{f_4}$: $\mathcal{A}_m$ and h_1 produce a block on $\mathcal{C}_1$ and no miner on $\mathcal{C}_2$ produces a block. $\mathcal{A}_m$ sends block to h_2 and maintains a perfect split. Event probability is $\alpha_1\alpha_2(1-\beta_1)(1-\beta_2)$.
- $\mathbf{f_5}$: h_1 produces a block on $\mathcal{C}_1$ and either h_2 or h_3 produce a block on $\mathcal{C}_2$. $\mathcal{C}_1$ and $\mathcal{C}_2$ persist (perfect split exists) and $\mathcal{A}_m$ mines on $\mathcal{C}_1$. Event probability is $\alpha_2\beta_1(1-\alpha_1)(1-\beta_2) + \alpha_2\beta_2(1-\alpha_1)(1-\beta_1)$.
- $\mathbf{f_6}$: h_1 produces a block on $\mathcal{C}_1$ and both h_2 or h_3 produce a block on $\mathcal{C}_2$. Three chains form $(\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$. $\mathcal{A}_m$ receives block from h_1 and both mine on $\mathcal{C}_1$. h_2 and h_3 mine on $\mathcal{C}_2$ and C_3. Event probability is $\alpha_2\beta_1\beta_2(1-\alpha_1)$.
- $\mathbf{f_7}$: Both $\mathcal{A}_m$ and h_2 produce blocks on $\mathcal{C}_1$ and either h_2, or h_3, or both produce blocks on $\mathcal{C}_2$. Three or four branches can appear. $\mathcal{A}_m$ mines with h_1 to maintain the hash rate advantage. Event probability is $\alpha_1\alpha_2\beta_1(1-\beta_2)+\alpha_1\alpha_2\beta_2(1-\beta_1)+\alpha_1\alpha_2\beta_1\beta_2$.
- $\mathbf{f_8}$: Both h_2 and h_3 produce blocks on $\mathcal{C}_2$ and no miner on $\mathcal{C}_1$ produces a block. $\mathcal{C}_1$ resolves and $\mathcal{C}_2$ and $\mathcal{C}_3$ form. $\mathcal{A}_m$ mines on h_2's branch for higher hash rate advantage. Event probability is $\beta_1\beta_2(1-\alpha_1)(1-\alpha_2)$.
- $\mathbf{f_9}$: No miner produces block on either $\mathcal{C}_1$ or $\mathcal{C}_2$. The original fork persists. Event probability is $(1-\alpha_1)(1-\alpha_2)(1-\beta_1)(1-\beta_2)$.

Fork Resolves:

- $\mathbf{r_1}$: h_1 produces a block on $\mathcal{C}_1$ before $\mathcal{A}_m$, and neither h_2 or h_3 produce a block on $\mathcal{C}_2$. $\mathcal{C}_2$ dissolves and no fork remains. Event probability is $\alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2)$.
- $\mathbf{r_2}$: Either h_2 or h_3 produce a block and no miner on $\mathcal{C}_1$ produces a block. Fork resolves and $\mathcal{A}_m$ mines on h_2's branch to maintain the hash rate advantage. Event probability is $\beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1)$.

Figure 9. Block race after algorithm 2. For each event, we show the event probability and $\mathcal{A}_m$'s strategy for the next round.

In other words, after algorithm 2, the next block from $\mathcal{C}_1$ is mined at t_o, and respectively at $t_o^{'}$ for $\mathcal{C}_1$. Therefore, the probability that $\mathcal{C}_1$ succeeds in producing the block before $\mathcal{C}_2$ becomes $t_o/(t_o + t_o^{'}) = \alpha/(\alpha+\beta)$ [20], [21], [32]. Similarly, the probability that $\mathcal{A}_m$ mines the next block on $\mathcal{C}_1$ before h_1 is $\alpha_1/(\alpha_1+\alpha_2)$, and the probability that h_1 mines the next block on $\mathcal{C}_1$ before $\mathcal{A}_m$ is $\alpha_2/(\alpha_1+\alpha_2)$. This analysis can be trivially extended to h_1 and h_2 on $\mathcal{C}_2$.

After $\mathcal{A}_m$ executes algorithm 2, $\mathcal{A}_m$ needs to maintain the fork for k consecutive blocks to violate $\mathcal{Q}_{cp}$. However, if the fork resolves and the resulting chain has more blocks than $100\alpha_1$ (*i.e.,* out of 100 blocks, more than 26 mined by $\mathcal{A}_m$), $\mathcal{Q}_{cq}$ is violated. Note that since there are two public chains, if the fork resolves before k, and $\mathcal{C}_1$ is the winning chain, $\mathcal{Q}_{cq}$ is violated even if $\mathcal{Q}_{cp}$ is preserved. Considering these situation, in the following, we concretely specify the race conditions under which the *HashSplit* attack succeeds or fails.

1) If forks persist more than k blocks, $\mathcal{Q}_{cp}$ is violated. The attack succeeds partially.
2) If forks resolve before k blocks and $\mathcal{C}_1$ is the winning branch, $\mathcal{Q}_{cq}$ is violated. The attack succeeds partially.
3) If forks persist for k blocks and resolve at $k+1$ block with $\mathcal{C}_1$ as the winning branch, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated. The attack succeeds completely.
4) If forks persist for k blocks and resolve at $k+1$ block, with all k blocks mined by $\mathcal{A}_m$, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated. Additionally, *HashSplit* attack becomes a majority attack. In a synchronous network, this probability is 0.0003 with $\alpha_1 = 0.26$ [32].
5) If forks resolve before or after k blocks and $\mathcal{C}_2$ is the winning branch, $\mathcal{A}_m$ loses all blocks, and the attack fails.

Clearly, *HashSplit* relies on block race outcomes in which forks persist or resolve. In Figure 9, we formally analyze all outcomes of a block race along with their probability distribution and $\mathcal{A}_m$'s strategies. We define a random variable X that specifies the probability distribution of block race outcome in Figure 9. We further define $\mathbf{F}$ and $\mathbf{R}$ as the sum of events in which forks persist or resolve. In (1) and (2), we show the probability $\mathbb{P}[X=\mathbf{F}]$ and $\mathbb{P}[X=\mathbf{R}]$.

$$\begin{aligned}\mathbb{P}[X=\mathbf{F}] &= \alpha_1(1-\alpha_2)(1-\beta_1)(1-\beta_2) + \alpha_1\beta_1(1-\alpha_2)(1-\beta_2) \\ &+ \alpha_1\beta_2(1-\alpha_2)(1-\beta_1) + \alpha_1\beta_1\beta_2(1-\alpha_2) + \alpha_1\alpha_2(1-\beta_1) \\ &(1-\beta_2) + \alpha_2\beta_1(1-\alpha_1)(1-\beta_2) + \alpha_2\beta_2(1-\alpha_1)(1-\beta_1) \\ &+ \alpha_2\beta_1\beta_2(1-\alpha_1) + \alpha_1\alpha_2\beta_1(1-\beta_2) + \alpha_1\alpha_2\beta_2(1-\beta_1) \\ &+ \alpha_1\alpha_2\beta_1\beta_2 + \beta_1\beta_2(1-\alpha_1)(1-\alpha_2) + (1-\alpha_1)(1-\alpha_2) \\ &(1-\beta_1)(1-\beta_2). \\ \mathbb{P}[X=\mathbf{F}] &= 3\alpha_1\alpha_2\beta_1\beta_2 - 2\alpha_1\alpha_2\beta_2 - 2\alpha_1\beta_1\beta_2 - 3\alpha_2\beta_1\beta_2 - 2\alpha_1\alpha_2\beta_1 \\ &+ \alpha_1\beta_2 + 2\alpha_2\beta_2 + 2\beta_1\beta_2 + \alpha_1\alpha_2 + \alpha_1\beta_1 + 2\alpha_2\beta_1 - \beta_2 \\ &- \alpha_2 - \alpha_1 + 1 \end{aligned} \tag{1}$$

$$\begin{aligned}\mathbb{P}[X=\mathbf{R}] &= \alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) + \beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) \\ &+ \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1) \\ \mathbb{P}[X=\mathbf{R}] &= 2\alpha_1\alpha_2\beta_1 + 2\alpha_1\alpha_2\beta_2 + 2\alpha_1\beta_1\beta_2 - 3\alpha_1\alpha_2\beta_1\beta_2 + 3\alpha_2\beta_1\beta_2 \\ &= -\alpha_1\alpha_2 - \alpha_1\beta_1 - \alpha_1\beta_2 - 2\alpha_2\beta_1 - 2\beta_1\beta_2 + \alpha_2 + \beta_1 + \beta_2 \end{aligned} \tag{2}$$

Plugging the hash rate of each miner from our threat model, $\mathbb{P}[X=\mathbf{F}]$ and $\mathbb{P}[X=\mathbf{F}]$ resolve to 0.6892 and 0.3108, respectively. From these values and Figure 9, we make the following conclusions.

1) With algorithm 2 as the starting point of a block race, there is a higher probability that the given fork persists or new forks appear. This favors the violation of $\mathcal{Q}_{cp}$.
2) The probability that a fork is resolved by an honest miner on $\mathcal{C}_1$ is $\alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) = 0.1275$. This is significantly than 0.6892 and favors the violation of $\mathcal{Q}_{cq}$.
3) The probability that a fork is resolved by any honest miner on $\mathcal{C}_2$ is $\beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1) = 0.2401$ (significantly less than 0.6892). This is the failure probability for *HashSplit*.
4) If there are M miners, potentially M branches can appear after a block race, albite with a negligible probability $\left(\prod_{i=1}^{|M|} h(i)\right)$. More branches increase the probability of violating $\mathcal{Q}_{cp}$. We show in Figure 9, how $\mathcal{A}_m$ deals with more than two branches.
5) Block race can be modeled as a state machine in which the outcome can be a fork with probability $\mathbb{P}[X_k=\mathbf{F}]$, or single branch with probability $\mathbb{P}[X_k=\mathbf{R}]$ [15], [24]. In Figure 10, we present a state machine in which S_0

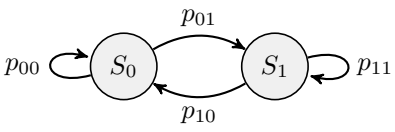


Figure 10. State machine representation of a block race. Transition probabilities are p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X = \mathbf{F}]$, $\mathbb{P}[X = \mathbf{R}]$, $\mathbb{P}[X = \mathbf{F}]$, and $\mathbb{P}[X = \mathbf{F}]$, respectively.

denotes a state of forks and S_1 denotes a state of forks resolved. The state transition probabilities p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X = \mathbf{F}]$, $\mathbb{P}[X = \mathbf{R}]$, $\mathbb{P}[X = \mathbf{F}]$, and $\mathbb{P}[X = \mathbf{F}]$, respectively.

6) Using Figure 10 and incorporating block delay from our dataset, we can compute the long term probability of a forked blockchain that violates $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

Incorporating Propagation Delay Advantage. Before computing the stationary distribution of Figure 10, it is important to incorporate $\mathcal{A}_m$'s mining advantage due to block propagation delay and block withholding. For instance, in $\mathbf{f}_1$, when $\mathcal{A}_m$ produces a block and withholds until h_2 or h_3 produce blocks, $\mathcal{A}_m$ can leverage the waiting time and the block propagation time to extend the newly mined block. The gap between $t_{a,1}$ and $t_{h,1}$, or $t_{a,2}$ and $t_{h,2}$ provides extra time for $\mathcal{A}_m$ to mine the next block. To model this advantage, first we need to characterize the effect of block propagation delay on each miner's hash rate. Let $t_{a,0}$, $t_{h_1,0}$, $t_{h_2,0}$, $t_{h_3,0}$ be times at which $\mathcal{A}_m$, h_1, h_2, and h_3 mine blocks with their hash rates α_1, α_2, β_1, and β_2, respectively. The relationship between block propagation delay and the hash rate can be obtained as follows.

$$\alpha_1 = \frac{\tau}{t_{a,0}}, \alpha_2 = \frac{\tau}{t_{h_1,0}}, \beta_1 = \frac{\tau}{t_{h_2,0}}, \beta_2 = \frac{\tau}{t_{h_3,0}} \tag{3}$$

$$\alpha_1 = \frac{\tau}{t_{a,0} + t_{a,1}}, \quad \alpha_2 = \frac{\tau}{t_{h_1,0} + t_{h,1}}$$
$$\beta_1 = \frac{\tau}{t_{h_2,0} + t_{h,1}}, \quad \beta_2 = \frac{\tau}{t_{h_3,0} + t_{h,1}} \tag{4}$$

Considering $\alpha_1 = 0.26$, $\alpha_2 = 0.25$, $\beta_1 = 0.25$, $\beta_2 = 0.24$, and $\tau = 600$ seconds, from (3), $t_{a,0}$, $t_{h_1,0}$, $t_{h_2,0}$ become 2308, 2400, 2400, and 2500, respectively. Plugging these values in (4), the hash rate of each miner becomes $\alpha_1 = 0.259$, $\alpha_2 = 0.244$, $\beta_1 = 0.244$, and $\beta_2 = 0.235$. Next, to incorporate $\mathcal{A}_m$'s advantage in a block race, we convert δ_1 in algorithm 1 as the mining advantage that increases α_1. In our model, δ_1 with respect to all honest miners is $52 - 2 = 50$ seconds. In 50 seconds, $\mathcal{A}_m$ gets $(\delta_1/\tau = 50/600) = 0.0833$ fraction of extra mining power over all other miners. As a result, the *effective hash rate* of each miner becomes $\alpha_1 = 0.3423, \alpha_2 = 0.2163, \beta_1 = 0.2163, \beta_2 = 0.2073$. Additionally, $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$ become 0.739 and 0.261. Logically, this advantage can be extended to miners when they resolve forks ($\mathbf{r}_1$ and $\mathbf{r}_2$ in Figure 9). If the fork resolves, the probability that a fork appears in the next round will be less than 0.739. More precisely, the winning miner will have $t_{a,1}$ advantage over $\mathcal{A}_m$, and $t_{h,1}$ advantage over other miners. Empirically, $t_{a,1}$ accounts for $2/600 = 0.0033$ and $t_{h,1}$ accounts for $52/600 = 0.087$ fraction of the mining power. Therefore, if the fork resolves, the probability that a fork appears in the next round becomes $\mathbb{P}[X = \mathbf{F}] = 0.683$. Using these values, we can construct the transition probability matrix for Figure 10.

$$P = \begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \left\| \begin{array}{cc} p_{00} & p_{01} \\ p_{10} & p_{11} \end{array} \right\| \end{array} = \begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \left\| \begin{array}{cc} 0.739 & 0.261 \\ 0.683 & 0.317 \end{array} \right\| \end{array}$$

In (5), we derive the stationary distribution of P to calculate the long term probability of a forked blockchain. The stationary distribution of P is a vector ψ such that $\psi P = \psi$.

$$0.739\psi_1 + 0.261\psi_2 = \psi_1,\ 0.683\psi_1 + 0.317\psi_2 = \psi_2,\ \psi_1 + \psi_2 = 1 \tag{5}$$

From (5), we get $\psi_1 = 0.724$ and $\psi_2 = 0.276$, showing that the long term probability of a forked chain is significantly greater than the probability of single branch. Using the stationary distribution, we can evaluate the impact of *HashSplit* on $\mathcal{Q}_{cp}$, $\mathcal{Q}_{cq}$, and the majority attack.

Common Prefix Property. Our analysis reveals for any block race of length k, $\mathcal{Q}_{cp}$ is violated ($\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ for any k) with 0.724 probability. For $k = 6$, P^6 yields $\mathbb{P}[X = F] = 0.72$. Therefore, *HashSplit* violates $\mathcal{Q}_{cp}$ with a high probability.

Chain Quality Property. We note in (4), block propagation affects the hash rate of each miner. As such, even if $\mathcal{A}_m$ does not partition the blockchain, it can still mine more blocks that its hash rate. For instance, assume an honest block race in our model. Since $\delta_1 = 50$ seconds, $\mathcal{A}_m$ has $50/(3 \times 600)$ fraction of mining advantage over the other three miners ($\alpha_1 = 0.26, \alpha_2 = 0.223, \beta_1 = 0.223, \beta_2 = 0.213$. If 100 blocks are mined, $\mathcal{A}_m$ will mine 28.29 blocks. Viewed from the ideal-world functionality, $\mu - \mu^{'} = 2.29 \neq \epsilon$. $\mathcal{A}_m$ mines two blocks more than its hash rate. Therefore, $\mathcal{Q}_{cq}$ is violated.

Common Prefix and Chain Quality. To violate both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$, the condition is that a fork persists or resolves after k blocks, and $\mathcal{C}_1$ is the winning branch. Figure 9 shows that event $\mathbf{r}_2$ is the only outcome where forks resolve to $\mathcal{C}_2$ with probability 0.2401. We can analyze this by branching S_1 in Figure 10 into two states and calculating the stationary probability of the state in which $\mathcal{C}_2$ is the winning chain. We compute that probability as 0.167. Therefore, with probability $(1 - 0.167 = 0.833)$, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated.

Majority Attack. From Figure 9, a majority attack happens if (1) $\mathcal{C}_1$ is the winning branch after k rounds, and (2) all blocks on $\mathcal{C}_1$ are mined by $\mathcal{A}_m$. This happens if for $k-1$ rounds, one of the events $\mathbf{f}_1$, $\mathbf{f}_2$, $\mathbf{f}_3$, $\mathbf{f}_4$, $\mathbf{f}_7$, or $\mathbf{f}_9$ occurs, followed by event $\mathbf{f}_1$ on the k_{th} round. Similar to the analysis above, we can decompose this into a state machine where S_0 determines the probability of events $\mathbf{f}_1$, $\mathbf{f}_2$, $\mathbf{f}_3$, $\mathbf{f}_4$, $\mathbf{f}_7$, or $\mathbf{f}_9$, while S_1 determines the probability of $\mathbf{f}_5$, $\mathbf{f}_6$, $\mathbf{f}_8$, $\mathbf{r}_1$, or $\mathbf{r}_2$. From Figure 9, we compute p_{00}, p_{01}, p_{10}, and p_{11} as 0.663, 0.337, 0.576, and 0.424, respectively. For $k = 6$, the result is $(0.63 \times 0.342) = 0.2187$. Therefore, with 0.2187 probability, *HashSplit* allows $\mathcal{A}_m$ to launch a majority attack with only 26% hash rate. In *lock-step* synchronous or *non-lock-step* synchronous networks, the probability of successful majority attack with 26% hash rate is ≈ 0.0003 [32].

To summarize, *HashSplit* violates the blockchain safety properties with a high probability. Additionally, it significantly

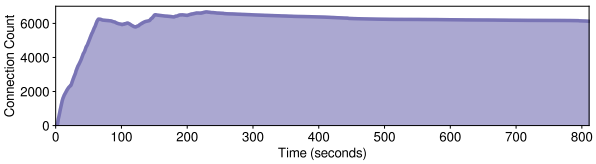
amortizes the requirement for the majority attack. In this paper, we have only presented an attack against the mining nodes. The attack can be launched against the non-mining nodes (*i.e.,* Bitcoin exchanges) to prevent them from generating $k-$confirmed transactions. As shown in our measurements §V, the non-mining nodes have relatively poor network synchronization compared to the mining nodes. Therefore, they are more vulnerable to *HashSplit*. As noted in §I, the concept of splitting the mining power to amortize the cost of the 51% attack is known in the literature [28], [18], [4]. However, these attacks require an adversary to disrupt the communication model which can be detected by the victims. In contrast, the *HashSplit* adversary does not disrupt the network communication, and only relies on the existing network latency and the mining policies to split the network. Therefore, the attack procedure goes undetected by the victims. In the last five years, 26% hash rate has been possessed by various mining pools, including BTC.com, Antpool, and F2Pool (see Antpool's example [1]). All these features make the *HashSplit* attack more practical, stealthy, and feasible in the current Bitcoin implementation.

VII. Attack Countermeasures

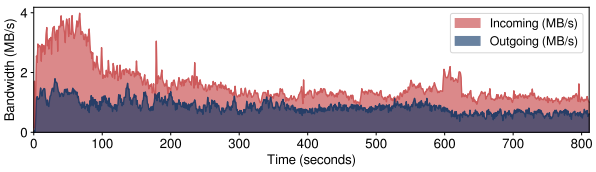
We also demonstrate the attack feasibility through discreet even-driven simulations. Due to space restrictions, we have moved simulation results to §F. In this section, we instead focus on attack countermeasures. Since *HashSplit* primarily exploits asynchronous network and block propagation delay, therefore, if $\delta_1 \dots \delta_4$ in algorithm 1 are minimized, $\mathcal{A}_m$: (1) cannot create a split between mining nodes, and (2) cannot leverage a significant mining advantage from propagation delay. Additionally, if all $P_i \in M$ form $M \times M$ topology, Bitcoin will closely reflect a *lock-step* or *non-lock-step* synchronous network, capable of countering *HashSplit*.

To minimize propagation delay and form $M \times M$ network topology, we have modified and released a Bitcoin Core version to counter the attack (anonymously released [2]). In our Bitcoin Core version, we have modified the source code to allow fast connectivity with all Bitcoin *reachable* nodes. Existing Bitcoin Core suffers from poor network reachability. A default node can take ≈120 days to connect to only 125 IP addresses (among the total 8K–10K) [35]. This is an unexpectedly long duration to connect with only a small fraction of the total network. To overcome this limitation, we have modified the Bitcoin Core source code and added scripts to enable fast connectivity in the *reachable* space. Details about new functionalities of our Bitcoin Core client can be found in [2].

For performance evaluation, we deployed our Bitcoin Core client on a local Linux machine and evaluated the connectivity speedup and bandwidth consumption. Results are reported in Figure 11, showing that our node connected with over $6K$ reachable nodes in less than 100 seconds. Moreover, the bandwidth consumption is under 6MB/s (4Mbps incoming and 2Mbps outgoing) during the initial connectivity phase. Once the number of connections stabilize, the bandwidth consumption reduces to ≈4Mbps. Our Bitcoin Core version



(a) Number of Bitcoin connections established in 800 seconds



(b) Incoming and outgoing bandwdith consumption in MBps.

Figure 11. Performance evaluation of our Bitcoin Core version deployed on a full node. In less than 100 seconds, our node connected with over $6K$ reachable nodes while maintaining the bandwidth overhead under 6Mbps.

is still in the testing phase and currently, it only supports fast connectivity to nodes running IPv4 or IPv6. Tor addresses can only be sequentially added which can take ≈45 minutes. In the future, we will add fast connectivity support for Tor nodes as well. From Figure 11, we note that a Bitcoin node can connect to all IPv4 and IPv6 mining nodes in less than 100 seconds. Through direct connectivity and better reachability, the node can instantly receive blocks from honest mining nodes thereby minimizing block delays and $\mathcal{A}_m$'s advantage. Even if a node is connected to all IPv4 and IPv6 nodes, $\mathcal{A}_m$ cannot achieve a perfect split unless all mining nodes start using Tor. However, we acknowledge that $M \times M$ topology does not fully counter the attack. Our measurements show that the network latency can be non-homogeneous such that two peers connected to a same node can have varying block propagation delay due to characteristics of the underlying Internet infrastructure (*i.e.,*, low bandwidth connection). Non-homogeneous latency can be measured by $\mathcal{A}_m$ to launch the *HashSplit* attack even in $M \times M$ topology. Therefore, in addition to network layer remedies, we also require application layer defenses to counter the attack.

For application layer defenses, we equip our Bitcoin Core with a *branch detection* mechanism. We note from Figure 9, during the attack, the victim nodes have multiple branches of same length in each round (*i.e.,* $\mathcal{C}_1$ and $\mathcal{C}_2$). Particularly, miners on $\mathcal{C}_1$ will continuously receive blocks from $\mathcal{A}_m$, immediately followed by blocks from other honest miners. We leverage this sequence of block reception to remove $\mathcal{A}_m$'s advantage and reduce the likelihood of a perfect split. In our modified node, we have enabled a *branch detection* mechanism in which if a node continuously receives $k = 6$ blocks from a same IP address, the node removes that connection and bans the IP address for twenty four hours. This means that $\mathcal{A}_m$ will lose a direct connection to all mining nodes and will not be able to achieve a perfect split. $\mathcal{A}_m$ may deploy Sybil nodes in the network to connect to the victim. However, in that case $\mathcal{A}_m$ will lose δ_1 advantage over the victim since the block will be first relayed to the Sybil and then to victim node. Therefore, a combination of high network reachability and *branch detection*

mechanism can mitigate the risk of *HashSplit* attack.

VIII. Discussion and Conclusion

***HashSplit*: Holistic Perspective.** The *HashSplit* attack is an outcome of rigorous theoretical analysis and systematic measurements for which we construct the Bitcoin ideal world functionality, identify the mining nodes, characterize block propagation patterns, and present the effects of asynchrony on the Nakamoto protocol in Bitcoin. We acknowledge that our measurement results can be further enumerated to curate new attack strategies for Bitcoin. However, the *HashSplit* attack is one demonstrative characterization of the execution model that is admissible in the computation model of Bitcoin.

Limitations and Future Work. We also acknowledge some limitations in our work. First, due to limited resources, we could not conduct the measurements for a longer duration that could have provided more interesting insights about the Bitcoin network. For instance, the per-second sampling of the network was highly storage-intensive, and we performed it to merely validate the asynchronous execution of Bitcoin which helped us to construct the threat model and the attack procedure. The second limitation of our work is overlooking the churn among the non-mining nodes. We observed that the IP addresses of the mining nodes remained static, while the IP addresses of the non-mining nodes frequently changed. In five weeks, we connected to over 36K IP addresses, and at any given moment, ≈10K IP addresses were reachable, including 359 addresses of the mining nodes. The non-mining nodes often changed their IP addresses which caused a churn in the network. We noticed that churn occurs due to cloud hosting. The users often turn off their nodes on cloud to save cost. Upon restarting, those nodes acquire new IP addresses. Since this behavior was observed among the non-mining nodes only, therefore, it did not affect the results for the *HashSplit* attack. Moreover, our proposed countermeasures provide a direction towards mitigating the attack by attempting to create a *non-lock-step* synchronous network. As in the case with all Bitcoin Core versions, it can be further improved to increase the network reachablity in less time.

Conclusion. Despite limitations, our overarching goal was to formulate the Bitcoin ideal functionality, identify the mining nodes, demonstrate weak network synchronization, and show the network asynchrony in the real word. The key takeway of our work is that the current Bitcoin network is fast moving towards an insecure space where existing notable attacks can be amortized and new attacks can be launched. We hope that our work bridges the gap between theory and practice while drawing more attention towards the large-scale measurements of Bitcoin and similar proof-of-work blockchain systems.

References

[1] "Antpoolhashrate." [Online]. Available: https://www.bitcoinmining.com/images/bitcoin-mining-pool-hash-rate-distribution.png

[2] Anonymous, "Improved bitcoin core to counter hash-split." [Online]. Available: https://anonymous.4open.science/r/56e77487-0470-4e10-b634-b13e939863c0/

[3] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: protecting bitcoin against routing attacks," in *Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA*, Feb 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/

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

[5] M. Bastiaan, "Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin," 2015, http://fmt.cs.utwente.nl/files/sprojects/268.pdf.

[6] BitcoinCommunity, "Bitcoin developer reference." [Online]. Available: https://bitcoin.org/en/developer-reference#p2p-network

[7] J. Carron, "Violin plots 101: Visualizing distribution and probability density." [Online]. Available: https://mode.com/blog/violin-plot-examples

[8] B. Community, "Bitcoin developer reference," 2018, https://bitcoin.org/en/bitcoin-for-developersn.com.

[9] ——, "Six confirmation practice in bitcoin," 2019, https://en.bitcoin.it/wiki/Confirmation.

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

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

[12] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *USENIX Security Symposium, Santa Clara, CA, USA*, N. Heninger and P. Traynor, Eds. USENIX Association, Aug 2019, pp. 801–818. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/das

[13] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proceedings of 13th International Conference on Peer-to-Peer Computing, IEEE P2P, Trento, Italy*, Sep 2013, pp. 1–10, https://doi.org/10.1109/P2P.2013.6688704.

[14] T. Duong, L. Fan, T. Veale, and H. Zhou, "Securing bitcoin-like backbone protocols against a malicious majority of computing power," *IACR Cryptology ePrint Archive*, vol. 2016, p. 716, 2016. [Online]. Available: http://eprint.iacr.org/2016/716

[15] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International Conference on Financial Cryptography and Data Security , FC, Christ Church, Barbados*, Mar 2014, pp. 436–454. [Online]. Available: https://doi.org/10.1007/978-3-662-45472-5_28

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

[17] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," *CoRR*, vol. abs/1801.03998, 2018. [Online]. Available: http://arxiv.org/abs/1801.03998

[18] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *SIGSAC Conference on Computer and Communications Security, Vienna, Austria*, Oct 2016, pp. 3–16. [Online]. Available: https://doi.org/10.1145/2976749.2978341

[19] S. Goldberg and E. Heilman, "Technical perspective: The rewards of selfish mining," *Commun. ACM*, vol. 61, no. 7, p. 94, 2018. [Online]. Available: https://doi.org/10.1145/3213006

[20] C. Grunspan and R. Pérez-Marco, "Double spend races," *CoRR*, vol. abs/1702.02867, 2017. [Online]. Available: http://arxiv.org/abs/1702.02867

[21] ——, "On profitability of selfish mining," *CoRR*, vol. abs/1805.08281, 2018. [Online]. Available: http://arxiv.org/abs/1805.08281

[22] J. Jang and H. Lee, "Profitable double-spending attacks," *CoRR*, vol. abs/1903.01711, 2019. [Online]. Available: http://arxiv.org/abs/1903.01711

[23] Y. Kwon, D. Kim, Y. Son, E. Y. Vasserman, and Y. Kim, "Be selfish and avoid dilemmas: Fork after withholding (FAW) attacks on bitcoin," in *SIGSAC Conference on Computer and Communications Security, CCS, Dallas, TX, USA*, Oct 2017, pp. 195–209. [Online]. Available: https://doi.org/10.1145/3133956.3134019

[24] Q. Li, Y. Chang, X. Wu, and G. Zhang, "A new theoretical framework of pyramid markov processes for blockchain selfish mining," *CoRR*, vol. abs/2007.01459, 2020. [Online]. Available: https://arxiv.org/abs/2007.01459

[25] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: bitcoin lightweight client privacy using trusted execution," in *USENIX Security Symposium, CA, USA*, N. Heninger and P. Traynor, Eds. USENIX Association, Aug 2019, pp. 783–800. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/matetic
[26] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes.(2015)," 2015.
[27] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.
[28] C. Natoli and V. Gramoli, "The balance attack or why forkable blockchains are ill-suited for consortium," in *International Conference on Dependable Systems and Networks, Denver, CO, USA*, June 2017, pp. 579–590. [Online]. Available: https://doi.org/10.1109/DSN.2017.44
[29] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *ACM SIGSAC Conference on Computer and Communications Security, CCS London, UK*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, Nov 2019, pp. 817–831. [Online]. Available: https://doi.org/10.1145/3319535.3354237
[30] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," *IACR Cryptology ePrint Archive*, vol. 2016, p. 454, 2016. [Online]. Available: http://eprint.iacr.org/2016/454
[31] L. Ren, "Analysis of nakamoto consensus," Cryptology ePrint Archive, Report 2019/943, 2019, https://eprint.iacr.org/2019/943.
[32] M. Rosenfeld, "Analysis of hashrate-based double spending," *CoRR*, vol. abs/1402.2009, 2014. [Online]. Available: http://arxiv.org/abs/1402.2009
[33] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on bitcoin: Colliding space, time, and logic," in *IEEE International Conference on Distributed Computing Systems ICDCS, Dellas, Texas, US*, July 2019.
[34] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security, FC, Christ Church, Barbados*, Feb 2016, pp. 515–532. [Online]. Available: https://doi.org/10.1007/978-3-662-54970-4_30
[35] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network," in *Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P)*, 2020.
[36] C. Wang, X. Chu, and Q. Yang, "Measurement and analysis of the bitcoin networks: A view from mining pools," *CoRR*, vol. abs/1902.07549, 2019.

APPENDIX

A. Proof of the Ideal World Functionality

In the following, we provide the proof for the ideal world functionality (§II and Figure 1).

Theorem 1 (Bitcoin Ideal World Functionality)**.** *If the protocol is run for l= 6 consecutive rounds, in which $k = 6$ blocks are produced, then with a high probability, $\mathcal{F}$ guarantees the common prefix property and the chain quality, as long as the adversary is bounded by $H/2$ hash rate.*

Proof. Prior to the proof, we present some practical considerations for our execution model. In the current Bitcoin protocol specifications, the average duration of a round is 10 minutes (600 seconds) and the parameter k for the common prefix is 6 blocks [9]. Moreover, Theorem 1 assumes that in each round, one block is produced. Therefore, for l consecutive rounds, a total of $k = l$ blocks are produced.

To prove Theorem 1, we assume by contradiction that the ideal world execution runs for $l = 6$ consecutive rounds after which $\mathcal{C}_1^{\lceil 6} \preceq \mathcal{C}_2$ does not hold. In other words, the two chains do not share a common prefix after pruning 6 blocks. For this to be true, in each round, at least two miners in M should concurrently produce a block at the same time t_0 and due to a fully connected or close to a fully connected graph, the remaining miners should receive the two blocks at t_1. As shown in Figure 1, the recipients toss a coin and select one of the two blocks (for generalization if x blocks are received, recipients roll x sided dice). The probability that for $l = k$ rounds, x blocks are concurrently produced is:

$$P(x|\lambda) = \left(\frac{e^{-\lambda}\lambda^x}{x!}\right)^k \tag{6}$$

Now assume a random variable X which represents an event that $\mathcal{C}_1^{\lceil 6} \not\preceq \mathcal{C}_2$ for $l = k$ rounds due to x concurrent blocks. And since each recipient has to roll an x sided dice if x blocks are received, therefore $P(X)$ (from (6)) becomes:

$$P(X) = \left(\frac{e^{-\lambda}\lambda^x}{x^2(x-1)!}\right)^k \tag{7}$$

With λ =1/600, $k = 6$, and $x = 2$, $P(X)$ is 0.00001. In other words, the ideal world functionality guarantees the common prefix for $k = 6$ with overwhelming probability of 0.99999.

To ensure the chain quality property, $\mathcal{F}$ specifies that no h_i for $P_i \in M$ has more than 50% hash rate. Otherwise, $\frac{h_i}{H}$ does not hold and $\mathcal{F}$ aborts. Moreover, in the winning chain, the number of blocks contributed by the honest miners is proportional to their hash rate. For instance, in a chain length of $l = 6$ rounds in which 6 consecutive blocks are produced, a miner with 14.3% hash rate should be able to contribute 1 block (μ_i). If a miner faithfully respects the protocol in Figure 1, its probability of contributing 1 block becomes $k\frac{h_i}{H}$. Plugging in the experimental values, the probability is 0.999 ($\mu_i^{'}$. Therefore, $\mu_i - \mu_i^{'}$ is 0.001. This is negligible (ϵ) as defined in the ideal world functionality Figure 1. □

In this section, we perform a comparative analysis of the two well-known models that characterize the functionality of Bitcoin. The first model is proposed by Garay *et al.* [16] that specifies the Bitcoin backbone protocol and provides the formal definitions of the "Common Prefix Property" and the "Chain Quality Property." The second model is proposed by Pass *et al.* [30] that analyzes the performance of Blockchain protocols in asynchronous networks.[3]

The Bitcoin Backbone Protocol [16]. Garay *et al.* assumed a synchronous network in which when a miner releases a block, it is received by all the other miners concurrently, with negligible delay. Therefore, in each round, all nodes execute the protocol in a *lock-step* [31]. Moreover, the model assumes $M = \mathbb{N}$, where the hash rate is uniformly distributed among all miners. Finally, the adversary does not control more than $|M|/2 = |\mathbb{N}|/2$ miners. In other words, the miner is bounded by 50% hash rate. Using these assumptions, the [16] proposes the two theorems for the common prefix property and the chain quality property.

Theorem 2. *(Common Prefix). In a typical execution, the common prefix property holds with a parameter $k \geq 2\lambda f$. Here, k is the number of blocks for the common prefix property,*

[3]Although Pass *et al.* call their model asynchronous, however, Ren *et al.* [31] show that their model is actually *non-lock-step* synchronous. For simplicity, in this section, we still refer to [30]'s model as asynchronous.

f is the probability that at least one honest miner produces a block, and $\lambda \geq 2/f$ is the security parameter.

Theorem 3. *(Chain Quality). In a typical execution, the common prefix property holds with a parameter $l \geq 2\lambda f$.*

Although the Bitcoin backbone protocol in [16] formally specifies the properties of the Bitcoin system, however, it makes some assumptions that deviate from the real world implementation. In the following, we briefly discuss them.

(1) Firstly, the model assumes $M = \mathbb{N}$ and the mining power to be uniformly distributed. However, as shown in §IV, $M << \mathbb{N}$ and the mining power is not uniformly distributed. (2) Secondly, the synchronous execution assumes that in each round, the block experiences no propagation delay. As a result, the adversary gains no advantage from the propagation delay. In our measurements, we have observed that the Bitcoin network is a sparsely connected graph. As such, even if we reduce the Bitcoin network to $M \times M$, the network may still not be synchronous. (3) Finally as we show in the *HashSplit* attack, the asynchronous network empowers the adversary to violate the common prefix property and the chain quality property with as low as 26% hash rate. Translated to the formulation of [16], this means that if the adversary controls $\approx |M|/4$ or $|\mathbb{N}|/4$ miners, the common prefix property and the chain quality property will not hold.

Blockchains in *non-lock-step* Synchronous Networks [30]. Improving the model of Garay *et al.* [16], Pass *et al.* [30] proposed an *non-lock-step* synchronous model to evaluate Bitcoin. Their proposition introduces a network delay parameter Δ that an adversary can add during block propagation. By the time the block reaches other miners, the adversary leverages Δ to gain a head start towards computing the next block. In the following, we analyze the model proposed in [30].

(1) The primary assumption of their model is that an adversary is able to compute a block, delay its transmission by an upper bound Δ, and broadcast it to the network. After the broadcast, all participants receive the block and the *non-lock-step* synchronous network starts to *emulate* the *lock-step* synchronous network [16]. Therefore the key difference in the *non-lock-step* synchronous model [30] and the *lock-step* synchronous model [16] is Δ, after which both models *emulate* the same behavior. (2) Δ gives an advantage to the adversary over all the other participants. Roughly speaking, in the Δ time window, the adversary is able to mine on top of its previous block. Moreover, [30] assumes that during Δ, all the other participants remain idle. More formally, the model specifies $\alpha \approx p(1-\rho)n$ to be the probability that an honest player computes the block. Here, p is the mining hardness and ρ is the fraction of nodes in n that the adversary controls. Moreover, $\beta = \rho np$ is the expected number of blocks that an adversary can mine in a round. (3) Once the bounded delay Δ is plugged into the system, the model in [30] assumes a parameter $\delta > 0$ such that to meet consistency property, the model has to satisfy $\alpha(1-(2\Delta+2)\alpha) \geq (1+\delta)\beta$. As long as $\rho < 0.5$ and $p < \frac{1}{pn\Delta}$, consistency is satisfied. (3) The bounded delay Δ also "*discounts*" the capability α of honest nodes to produce blocks in a round. To formally capture that, the model in [30] introduces $\gamma = \frac{\alpha}{1+\Delta\alpha}$ which is the discounted version of α or the effective mining power gained by delaying the block propagation by Δ.

Using the aforementioned assumptions and a security parameter κ, [30] proposes two theorems to characterize Nakamoto consensus specific to the Bitcoin operations.

Theorem 4. *(Chain quality).* For all $\delta > 0$ any $p(.)$, $(\prod^{p}_{nak}, \mathcal{C}^{p}_{nak})$ has the chain quality*:*

$$\mu^{p}_{\delta}(\kappa, n, \rho, \Delta) = 1 - (1+\delta)\frac{\beta}{\gamma} \tag{8}$$

Theorem 5. *(Consistency).* Assume there exists some $\delta > 0$ such that

$$\alpha(1-(2\Delta+2)\alpha) \geq (1+\delta)\beta \tag{9}$$

Then, except with exponentially small probability (in T), Nakamoto consensus satisfies T-consistency, assuming the network's latency is bounded by Δ.

The consistency property in (9) can also be interpreted as the common prefix property in [16]. The T-consistency specifies that the two ledgers must share a common prefix after pruning T blocks from their chains ($\mathcal{C}_1^{\lceil T} \preceq \mathcal{C}_2$). Moreover, in [30] (Section 3.5), the authors mention "for instance, in the Bitcoin application, we are interested in achieving T-consistency for T = 6." This is similar to our formulation of the ideal world functionality and its proof, where we prove that ($\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, for k = 6). However, in the *HashSplit* attack, we show that the adversary violates the consistency property by deviating from the ideal world functionality. Additionally, we provide Theorem 6 to improve the characterization of [30], where an *non-lock-step* synchronous model starts emulating the *lock-step* synchronous model.

Experimental Interpretation. The experimental setup in [30] assumes a network in which each node has the capability of producing a block. Moreover, it also assumes $n = 10^5$, $\Delta = 10s$, and the diameter of the network to be less than 10 hops. The results show that Nakamoto consensus tolerates a 49.75% attack, under $\Delta = 10s$ bound.

1) Non-lock-step *Synchronous Model in Real World:* Although the *non-lock-step* synchronous model in [30] enhances the understanding about the Bitcoin system, however, it also makes some generalized assumptions that may not reflect the actual Bitcoin system as we largely observe in our experiments (§C). Firstly, [30] assumes that during Δ, other mining nodes remain idle, and after Δ, the system abruptly starts emulating *lock-step* synchronous behavior. This would imply that when an adversary delays a block by Δ, no other miner receives the block in the meantime, and after Δ, all miners receive the block instantly and start mining on top of it. This generalization does not capture the real world Bitcoin operations. For instance, assume there are $w = |M|$ mining nodes in the system. Each node is $1, ..., w$ hops away from a typical mining node (adversary in this case). Further, assume that the adversary releases the block and the block incurs a delay at each hop. Now, $P_1 \in M$ receives the block after Δ_1, $P_2 \in M$ receives the block after Δ_2, and $P_w \in M$ receives the block

after Δ_w. Naturally, $\Delta_1 < \Delta_2 < \Delta_w$. In PoW, each miner is motivated by its interests. If the miner receives a new block, for which it has been unsuccessfully mining, it immediately drops its computation and starts mining on top of the newly received block. Therefore, the network does not exhibit synchronous behavior until all $|M|$ miners receive the block. Moreover, if by the time $P_w \in M$ receives the block, and another miner $P_1 ... P_{w-1} \in M$ produces the next block, then the system may never exhibit the synchronous model. For a synchronous execution, all miners in M should be mining for the same block. Realizing this, below we present a theorem that puts a stronger bound on the emulation of *lock-step* synchronous model. We also provide a proof sketch.

Theorem 6. *Bitcoin emulates synchronous behavior iff each $P_i \in M$ receives b_r before another $P_j \in M$ produces b_{r+1}.*

Proof. Assume by contradiction that a miner $P_i \in M$ receives blocks b_r at time t_1 and another miner $P_j \in M$ produces another block b_{r+1} at t_0 where $t_0 < t_1$. $P_j \in M$ releases its block and a subset of miners $M_1 \in M$, where $P_i \notin M_1$ start mining on top of b_{r+1}. Since P_i has not received b_{r+1} so it will continue to mine on top of the received b_r. As a result, not all miners in M are solving for the same PoW (*i.e.,* extending the same block). Hence they do not exhibit synchronous behavior. □

Also note that by the time $P_w \in M$ receives the b_r, other miners $P_1 ... P_{w-1}$ will not be in an "idle" state, as assumed in [30]. They will be mining on top of the block b_r. Therefore values of α, γ, and δ in Theorem 4, Theorem 5 will change. In the case that we have outlined above, α will become $\alpha_1, ..., \alpha_w$, and similarly δ and γ would change to $\delta_1, ..., \delta_w$ and $\gamma_1, ..., \gamma_w$, respectively. The value of β will however remain unchanged. As a result, in the real world settings, the overall advantage of the adversary, due to Δ will decrease. Plugging this into [30], the chain quality in the honest environment actually becomes $1 - [(1+\delta_1)\frac{\beta}{\gamma_1} \times (1+\delta_2)\frac{\beta}{\gamma_2} \times, ..., \times (1+\delta_w)\frac{\beta}{\gamma_w}]$.

Other assumptions in the experimental interpretation of the model are the network size of 10^5 nodes and $\Delta = 10s$. The paper uses the network size as 10^5 nodes in order to support the real world Bitcoin hash rate. However, they assumed that the hash rate is uniformly distributed among all nodes. As a result, their experimental results matched the ones presented by Decker *et al.* [13]. However, as we have already shown through our measurements, the network size is significantly less than 10^5 nodes and the mining power is not uniformly distributed. As such, the *non-lock-step* synchronous model can be improved to characterize the actual Bitcoin system.

B. Key Challenges in Data Collection

The first challenge was to enable our crawlers to connect with thousands of reachable Bitcoin nodes. A Bitcoin node is reachable if it accepts incoming connections from other nodes [10]. Unreachable nodes do not accept incoming connections mostly due to reaching the maximum connectivity limit. The default limit is 125 connections which can be increased by modifying the *Bitcoin.conf* file. In our initial observations made through Bitnodes, we noticed that Bitcoin nodes can have IPv4, IPv6, and Tor addresses [10]. Implementation of each also requires modifications in *Bitcoin.conf*.

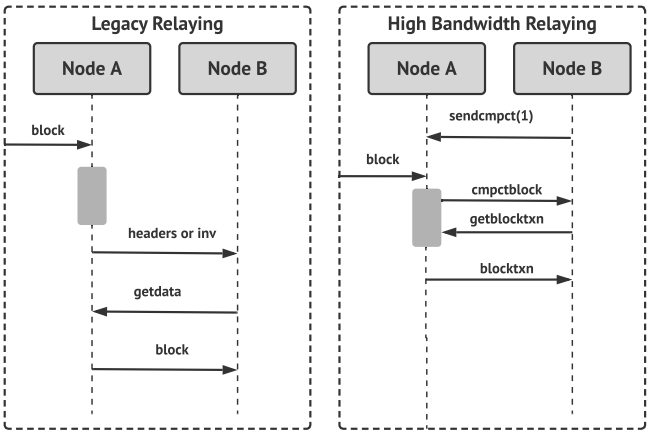


Figure 12. Changes in the Bitcoin communication model, originally proposed in BIP-152 [11]. In the legacy relaying, when Node A receives a block, first it verifies the contents of the block, then issues an inventory message, to which Node B responds with a *getdata* message. In the high bandwidth relaying, however, first Node B sends a *sendcmpct* message with first Boolean set to 1. When Node A receives a block, it immediately forwards it to Node B without validating headers. Node B reconstructs the block from its mempool.

Another challenge was to collect data from nodes that we were connected to and identify the mining nodes. The prior work [26], [13], [4] used the INV messages to locate the mining nodes. Their technique was to identify the node that first announces the INV message corresponding to a block. The first node would naturally be the mining node. This method has two shortcomings. Firstly, due to recent changes in Bitcoin Core implementation, some nodes do not advertise INV message before releasing the block [11]. Instead, they exchange a CMPCTBLOCK message at the connection handshake, and when they have a block, they simply forward it without advertising through INV message (see Figure 12 for details). Secondly, saving all INV messages from the network can be storage-intensive, especially when connected to thousands of nodes. Therefore, we needed techniques to compactly store the data with a smaller footprint.

C. Addressing Challenges

In this section, we show how we systematically addressed each challenge. We begin by providing insights into the Bitcoin Core software client that is installed on a node to connect with the Bitcoin network. The Bitcoin Core directory contains files that maintain the information about the blockchain and the user wallet. However, relevant to this work, we will only focus on two important files, namely the configuration file called *Bitcoin.conf* and the peers file called *peers.dat*.

***Bitcoin.conf* File.** The configuration file specifies the runtime options when the application is launched. [4] The file contains a set of command line instructions that are executed when a graphical interface (*Bitcoin-qt*) or the headless interface

[4]**Addendum: DHCP Configuration** To run a Bitcoin full node, the host machine must be assigned a static internal IP address by configuring the Dynamic Host Configuration Protocol (DHCP) on the router [8]

```
0000000050  0000 0000 0000 0000 0000 0000 ffff 4647  ..............@.
0000000060  f111 208d fd6b 88c0 8724 95c1 be86 064d  ..k...$.....M..
0000000070  0b90 2473 0000 0000 0000 0000 0000 0000  $s............t.
0000000080  7498 0200 405d c35c 0d04 0000 0000 0000  ...;.\..........
0000000090  0000 0000 0000 0000 0000 ffff 0dd1 40e3  ..........h... .
00000000a0  208d fd6b 88c0 8724 95c1 be86 064d 0b90  .k...$.....M..$s
```

Figure 13. Sample output of *peers.dat* file viewed in the hex. The hex value "208d," in green corresponds to 8333 in decimal, which is the default Bitcoin port. The preceding 4 Bytes show the IP address."46 47 f1 11 20"in hex is "70.71.241.17" in decimal.

(*bitcoind*) are launched. For our study, since we expected to connect with thousands of nodes, we had to change the default setting of "*maxconnections*" from 125 to a large number (*i.e.,* 100,000). Next, we set "*server =1*" in the configuration file so that *bitcoind* may function as a HTTP JSON-RPC server [6]. We also set the username and password for the JSON-RPC server which was later used by our client, programmed in NodeJS, to communicate with the server. Although there is a list of RPC commands that provide the state information of the Bitcoin server and the blockchain, however, our study is primarily concerned with five commands, namely *addnode*, *getconnectioncount*, *getchaintips*, *getpeerinfo*, and *getblockchaininfo*. As the names suggest, *addnode* creates a connection with a new node, *getconnectioncount* shows the count of all the connected nodes, *getblockchaininfo* provides information about the best chain with the longest prefix, *getchaintips* displays one or more local chains, and *getpeerinfo* provides information about all the connected nodes, including their latest block, the time of data exchange etc. For data storage optimization, we used the RPC API to compactly store the network information in the JSON format.

To maximize reachability, the *addnode* command is the highly useful. A typical execution of the command on the command line is "*bitcoin-cli addnode node address add | remove | onetry*. Here, *"node address"* is the address (IPv4, IPv6, or Tor), and *"add", "remove", and "onetry"* are the optional parameters that are passed with the command. The *"add"* parameter works only if the total connections of the node, issuing the command, are less than 125. Otherwise, *"onetry"* option is selected when a peer is already connected to more than 125 nodes. While connecting to a target node, if the target node's connection slots are full, it will refuse the connection. In that case, we repeatedly made connection requests using *"onetry"* after 60 seconds backoff period.

Peers.dat File. The *peers.dat* file compactly logs GETADDR and ADDR messages exchanged among nodes that are useful in obtaining the IP addresses of the *reachable* nodes [26]. A sample file output (viewed in hex) is shown in Figure 13.

We performed a manual file inspection, correlating its contents with the Bitcoin source code to extract the required information. The file structure of *peers.dat* is specified in *addrdb.cpp* consisting of header, data, and checksum. The comments in *addrdb.h* reveal the header's organization, including the file's data structures, format, and size. The file consists of table entries of known and the newly discovered IP addresses for the host node. The CAddrInfo structure in *addrdb.h* points to CSAddress in *protocol.h* which eventually

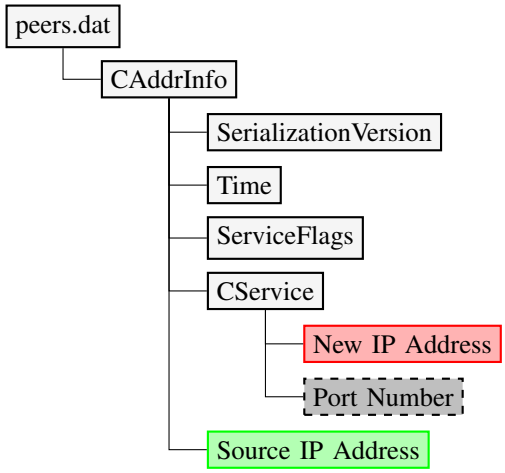


Figure 14. File structure of *peers.dat* file. The Source in green is the IP address to which our crawler has connected. New IP Address in red is the address to which the Source IP address has connected or received.

points to CService found in *netaddress.h*. By decoding the contents of these files, we extracted the IP addresses and port numbers sent by a node (source) during GETADDR and ADDR interactions with our crawlers. A truncated file structure is shown in Figure 14. The "Source IP Address" is the address of the node to which our crawler was connected and the "New IP Address" is the address of the node to which the "Source IP Address' either established a connection or learned from other connections. We obtained two useful insights from Figure 14: 1) new IP addresses to connect and maximize connectivity, and 2) the relationship between the Source IP Address and the New IP Address, which we use to construct the network topology. We developed a *peers.dat* parser for our crawlers.

In summary, we modified the *Bitcoin.conf* file to allow each crawler to connect with thousands of nodes. Moreover, to overcome the issues related to GETADDR interactions in order to obtain the knowledge about the network topology, we decoded the *peers.dat* file. The *peers.dat* file compactly logs the interactions between the GETADDR and ADDR messages. We decoded the *peers.dat* file (§C) to obtain the IP addresses of the reachable nodes in the Bitcoin network.

Minor Challenges. Along with the major challenges of data collection, there were minor issues related to bandwidth and file system management that are worth mentioning. We used high-speed fiber optic connection at each crawler to avoid unnecessary delay that would affect our measurements result. For instance, with a low-bandwidth internet connection, it is possible that a mining node might send a block to our node and another node concurrently. As such, due to low bandwidth, our crawler recievis the block from the other node instead of the mining node. While this remains a possibility, it is however unlikely. The reason is that even if the other node receives the block before our crawler, it will first verify the PoW correctness of the block and all the transactions in the block. It will then advertise or directly send the block to our crawler. The additional time of verifying the blocks and its transactions will be naturally less than the propagation delay during which

Table I
COMPARISON WITH PRIOR WORK: THIS TABLE IS INTENDED TO GIVE A SUCCINCT SUMMARY TO DISTINGUISH *HashSplit* FROM KNOWN RESULTS IN THEORETICAL AND EMPIRICAL CHARACTERIZATION OF THE BITCOIN BLOCKCHAIN. IN THE CONTEXT OF THE PRESENTED IDEAL WORLD FUNCTIONALITY, §A EXPLAINS THE THEORETICAL RESULTS IN DETAIL WHILE §E PROVIDES A PRECISE CHARACTERIZATION OF KNOWN ATTACKS ON THE BITCOIN IMPLEMENTATION.

		Execution Model	Network Synchronization	Majority Attack	Public Chains	Mining Nodes
Theoretical Models	Garay *et al.* [16]	*lock-step* synchronous	Good	51%	one	–
	Pass *et al.* [30]	*non-lock-step* synchronous	Good	49%	one	–
	Natoli *et al.* [28] (Ethereum)	*lock-step* synchronous	Good	5%	many	–
	HashSplit	asynchronous	Poor	26%	two	359
Measurements	Decker *et al.* [13]	*non-lock-step* synchronous	Weaker	49%	one	—
	Saad *et al.* [33]	*non-lock-step* synchronous	Poor	—	one	—
	Miller *et al.* [26]	*lock-step* synchronous	partially connected	—	one	100
	HashSplit	asynchronous	sparsely connected	26%	two	359

our crawler will directly receive the block from the mining nodes. Nevertheless, considering this to be a possibility, we used high-speed fiber-optic Internet connection to avoid even the smallest possibility of such an occurrence and receive the blocks directly from the mining node.

D. Cross Validating Heuristic 1

In order to validate the correctness of **Heuristic 1**, we performed a short experiment to check if the network information matched with the Bitcoin RPC API data. First, we deployed a crawler in Bitcoin and launched Bitcoin Core using the command *bitcoind -debug=1* and stored results in a separate file. The command *bitcoind -debug=1* collects the information exchange (at the network layer) between the crawler and the nodes to which it is connected. The information exchange, also includes the time at which a new block is received from a connected peer. Concurrently, we also deployed our RPC API crawling scripts to store the JSON output when a new block was received. We ran the experiment for 12 blocks (≈2 hours). After completion, we parsed the file and extracted the IP address of the peer that first advertised a new block header or directly sent the block following the CMPCTBLOCK method. We created a list of blocks and the corresponding IP address of the sending peer. Next, we parsed the JSON files obtained through RPC API and applied **Heuristic 1** and obtained the second list of blocks and IP addresses. We compared the two lists and found an exact match for all the blocks. This experiment validated the correctness of our methodology in detecting the mining nodes.

E. Notable Attacks on Bitcoin

In this section, we discuss the notable attacks on PoW-based blockchain systems. We will discuss the 51% attack, the selfish mining attack, and the double-spending attack.

51% Attack. The 51% attack is a classical weakness in blockchains where an adversary acquires a majority of the network's hash rate to gain control over the blockchain [15], [14]. The 51% attack primarily relies on the ability to generate the "longest chain" in the long run [5]. To understand how the majority attack works, assume an attacker with the hash rate h_a, participating in the block race. The attacker is s blocks behind the rest of the network and aims to catch up with a private chain that is longer than the public chain. If the rest of the network with hash rate $H - h_a$ finds the next block then the attacker will be $s+1$ blocks behind, with his success probability as a_{s+1}. Conversely, if the attacker finds the next block with probability, the gap will reduce by $s-1$, with his success probability as a_{s-1}. Given this information, a_s must satisfy the following recurrence relationship.

$$a_s = \frac{(H-h_a)a_{s-1}}{h_a} + \frac{(h_a)a_{s+1}}{H-h_a} \tag{10}$$

$$a_s = min\Big(\frac{h_a}{H-h_a}, 1\Big)^{max(s+1,0)} \tag{11}$$

$$a_s = \begin{cases} 1 & \text{if } s<0 \text{ or } h_a > (H-h_a) \\ \left(\frac{h_a}{H-h_a}\right)^{s+1} & \text{if } s>0 \text{ or } h_a < (H-h_a) \end{cases} \tag{12}$$

Note from above, if $h_a > (H - h_a)$ (the attacker has more than 50% hash power), it will succeed in the attack. As a result, the attacker will be able to have a strong control over the blockchain, depriving other miners from extending it.

Selfish Mining. Selfish mining is a form of block withholding attack, in which the adversary computes a block and does not publish it [32]. Instead, it keeps on extending its private chain in hopes to attain a longer chain than the competing public chain. When the adversary achieves that, it releases its longer private chain. In Bitcoin, nodes switch to the chain with the longer prefix, thereby invalidating the public chain computed by the honest miners. To evaluate the attack feasibility, we again refer to (10). If the adversary has 51% hash rate, its private chain will eventually be longer than the rest of the network, therefore guaranteeing a successful attack.

Double-spending. Double-spending or equivocation is when an attacker spends their cryptocurrency token twice [22]. The double-spending attack is launched in various ways. One possible method is that the attacker sends the transaction to a receiver and the receiver delivers a product before the transaction is confirmed. The attacker then sends the other transaction to himself. Both transactions are received by a miner, who can only accept one of them. Therefore, with 0.5 probability the recipient could be tricked. The other strategy could be that the attacker transacts with the recipient and the transaction gets confirmed in the public blockchain. The attacker then generates the other transaction, adds it to the private blockchain, and launches a selfish mining attack. If the selfish mining succeeds (with probability 1 if the attacker has 51% hash rate), then the recipient's transaction will be invalidated along with the public blockchain.

In each of the aforementioned attacks, the attacker's success is guaranteed if he has 51% hash rate and the network is

synchronous. If the hash rate is less than 51% or the network is asynchronous, the success probability decreases. For instance, assume a selfish mining attack in which the adversary is able to mine a private blockchain which is one block longer than the public blockchain. Next, the adversary releases the chain. Assuming a synchronous environment, in the next time step, the entire network will receive the attacker's chain and switch to it. However, if the network exhibits an asynchronous behavior such that the chain experiences propagation delay, then it is possible that any other honest miner is able to produce the block and propagate it faster. As a result, the selfish mining attack will not succeed in that case.

Another key aspect of these attacks is that there are two competing chains. One chain is the public chain on which honest miners work. The other is the private chain on which the attacker works. The private chain is kept hidden from the network until the attack is launched. However, a distinguishing aspect of the *HashSplit* attack is that there are two competing public chains in the network on which honest miners and the attacker are working concurrently. This situation is only possible in an asynchronous network where at one time, only a particular set of miners have visibility of the block. The adversary exploits this opportunity to launch the attack.

Blockchain Forks. When two or more conflicting chains exist in a blockchain system, it is called a fork. A fork essentially violates the common prefix property. In all the attacks mentioned above, when the attacker substitutes the public chain with his private chain, it first forks the public chain and violates the common prefix property. To resolve the fork, the network follows the longest prefix rule and switches to the longer chain with higher PoW behind it.

While a fork violates the common prefix property, the hash rate-based attacks violate the chain quality. The chain quality ensures that in the public ledger, the number of blocks contributed by a miner should be proportional to the miner's hash rate. In Bitcoin, the chain quality holds if the attacker's hash rate is below 51% (10). Exceeding the limit would give the attacker a permanent control over the chain growth. Significant to the *HashSplit* attack, we show that by mounting new attack strategies in the asynchronous network, the attacker can violate the chain quality with only 26% hash rate.

F. Simulations and Results

n the following, we demonstrate *HashSplit* through computer simulations. We developed a simulator in Python that implements the PoW algorithm. For simplicity, and to enable mining on a CPU, we significantly lowered the target of PoW. To perform concurrent mining, we used the *multiprocessing* library which effectively side-steps the "Global Interpreter Lock" by replacing threads with subprocesses. As a result, we were able to efficiently leverage the multi-core processor to simulate a block race among multiple mining nodes. For this experiment, we set up six miners, each with a *genesis* block and a block prototype containing dummy transactions. We assigned 26% processing power to the adversary and the remaining 74% randomly assigned to the other five miners.

For simulations, we created the network topology in a way that the adversary was directly connected with all five miners

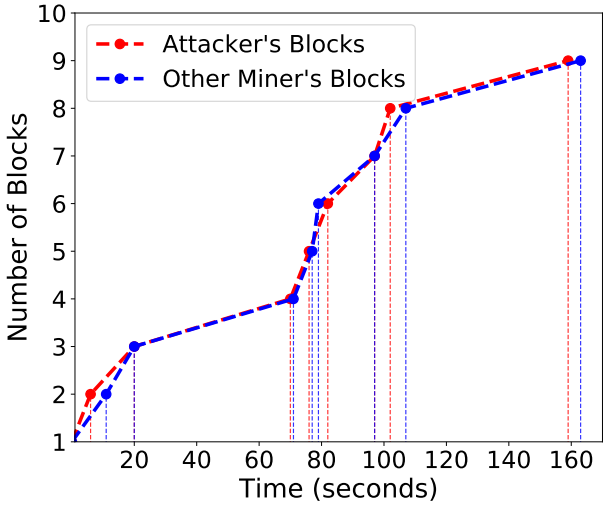


Figure 15. Simulation showing an execution of the *HashSplit* attack. In each round (except 5th), the adversary with 26% hash rate is the first to produce a block and follows algorithm 2. In the 5th round, the adversary manages to produce the block before $t_{h,2}$. Adversary releases the chain after 8th block

so that whenever a new block was produced by any node, the adversary directly received the block. Additionally, the topology among the other miners was adjusted to mimic the real-world Bitcoin network in which random delay affected the block propagation, thereby allowing the adversary to propagate two blocks among two separate sets of miners. We had two options to curate the network topology. One was to implement sockets and add deterministic delay in the block propagation. However, we noticed that socket implementation incurred significant processing overhead which wasted critical CPU cycles that could be utilized in solving the PoW. Again, favoring simplicity, we instead used an access control policy to construct the network topology. When a miner produced a block, the block was added to the public blockchain stored in a file. Next, the file sent the updated blockchain to each process (miner) of the execution. Based on the pre-determined relationship between the block producing miner and other miners, we introduced the deterministic delay in the blockchain broadcast. For instance, since adversary was directly connected to each miner, it immediately received the block when the blockchain was updated in the file. In contrast, if two miners were not directly connected to each other, a block produced by one was sent to the other after 100 milliseconds delay. This strategy allowed us to construct the network topology without incurring the overhead of a client-server socket implementation. However, since we will open-source our simulation setup in futrue, thereofre, it can be tailored to any custom topology implementation.

Figure 15 shows that except for the 5th block, the adversary was able to find a block before any other miner in the network. After computing the block, the adversary waited for any other miner from the competing chain to release the block. In the meantime, it continued extending its own chain atop its previously mined block. In our results, we observed that at the 5th block, a miner on the second public chain produced the block before the adversary. However, the adversary was

able to mine the block immediately after, and it released the block to keep the branch alive with $|M|/2$ miners. Finally, at the 8th block, when the adversary mined its block, it did not withhold it. Instead, it released the block to all miners in M, thereby forcing them to switch to the longer chain.

Our simulation results validated the theoretical propositions that by exploiting the asynchronous network, the adversary maintained to branches of the public blockchain to violate the common prefix property. The resulting chain had a majority of blocks mined by the adversary, which violated the chain qulaity.