HashSplit: Exploiting Bitcoin Asynchrony to Violate Common Prefix and Chain Quality

Muhammad Saad‡, Afsah Anwar‡, Srivatsan Ravi†, and David Mohaisen‡
‡University of Central Florida †University of Southern California
saad.ucf@knights.ucf.edu, afsahanwar@knights.ucf.edu, srivatsr@usc.edu, mohaisen@ucf.edu
03-05-2021

Contents

1 Introduction and Related Work 2

2 The Bitcoin Ideal World Functionality 4

3 Data Collection 7
  3.1 Bitcoin Peer-to-Peer Network ................................. 7
  3.2 Data Collection System ........................................ 8

4 Identifying the Mining Nodes 9
  4.1 Detection Technique and Results .............................. 9
  4.2 Revisiting Partitioning Attacks ............................. 10

5 Network Synchronization 11
  5.1 Revisiting Information Propagation .......................... 12
  5.2 Asynchrony in Bitcoin ....................................... 12

6 The HashSplit Attack 14
  6.1 Threat Model and Attack Objectives ........................ 15
  6.2 Attack Procedure ............................................. 15
    6.2.1 Identifying Vulnerable Nodes ............................. 15
    6.2.2 Blockchain Splitting ..................................... 15
    6.2.3 Block Race .............................................. 17

7 Attack Countermeasures 21

8 Discussion and Conclusion 22
  .1 Proof of the Ideal World Functionality ........................ 24

A Algorithmic Analysis of Bitcoin Consensus 24
  A.0.1 Non-lock-step Synchronous Model in Real World ........... 26
  A.1 Notable Attacks on Bitcoin .................................. 26
  A.2 Simulations and Results ..................................... 28
Abstract

The safety of the Bitcoin blockchain relies on strong network synchrony. Therefore, violating the blockchain safety requires strong adversaries who control a mining pool with \( \approx 51\% \) 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 in a synchronous network. 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 attacks. 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 discover non-uniform block propagation delay among the mining nodes showing 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, proposed attack, and countermeasures open new directions in the security evaluation of Bitcoin and similar blockchain systems.

1 Introduction and Related Work

Bitcoin is a dynamically evolving distributed system that has significantly scaled up in recent years [10]. As Bitcoin continues to grow and inspire other decentralized applications, its security features are continuously investigated using theoretical analysis and measurement techniques [23, 27]. 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 [25, 14, 28, 29] 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, 24].

To incorporate the mining centrality in a theoretical model, we construct the Bitcoin ideal functionality (§2), 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. [14], 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 [11]. To address this limitation, Pass et al. [28] extended the work in [14] and analyzed Bitcoin in the non-lock-step synchronous settings [29]. The non-lock-step synchronous model assumes a network in which all miners experience the 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 §1, we analyze the shortcomings of these models along with refinements to their assumptions. In §5.2, 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

---

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 [24, 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.
chain quality [14]. In §6, we show that the asynchronous network significantly relaxes the requirement to violate these two 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, 31, 15], (2) discovering influential nodes in mining pools [24, 4], and (3) measuring information flow and network synchronization [11, 31]. 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 [11]), and (3) a combination of the two attacks (i.e., spatio-temporal partitioning in [31] and the balance attack in [26]). In the context of measuring the information propagation in the Bitcoin network, the two related studies to our work are by Decker et al. [11] and Saad et al. [31].

In 2013, Decker et al. [11] conducted the first measurement study to analyze the information propagation in the Bitcoin 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. [31] 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 network size and suggested that the delay incentivizes malicious miners to create soft forks. In §4.2, 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 [14, 28] and ignore the critical distinction between the mining and non-mining nodes (i.e., in [31, 11, 26]). 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 [28]. 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 [26] 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. [16]). 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 [26]) 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
The Bitcoin ledger; the common prefix property and the chain quality property [14] (§2). 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 (§3). We develop heuristics to identify the mining nodes and identify 359 IP addresses of the mining nodes using those heuristics (§4).

3) We measure the block propagation patterns in the Bitcoin network (§5) 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 §5.2.

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 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 [2].

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 §8, and appendices with supplementary findings in §A.2.

2 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) [25]. 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 [34]. Since there are fewer mining pools than the number of Bitcoin users, therefore, there are fewer mining nodes in the network.

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 [14, 28], 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 [14]. In Appendix §A, we explain the model assumptions and the main theorems derived in [14, 28] by presenting the experimental interpretation of their results in the context of our ideal functionality.

First, we define \( N \) as the set of all reachable IP addresses of Bitcoin nodes. We define \( \text{VIEW}_i^C \) as the blockchain view of a single node \( P_i \in N \), where \( C \) is the blockchain ledger. The Bitcoin backbone protocol [14] states that the inter-arrival time between two blocks must be sufficiently large that each \( P_i \in N \) has \( \text{VIEW}_i^C \) (i.e., in ≈10 minutes, all \( P_i \in N \) have the up-to-date blockchain). Next, we define \( \{ M \subset N \} \) 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 \( 0 < h_i < 1 \). \( H = \sum_{i}^{\mid M \mid} h_i = 1 \) 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 \( Q_{cp} \), with parameter \( k \) specifies that for any pair of honest nodes \( P_1 \) and \( P_2 \), adopting the chains \( C_1 \) and \( C_2 \) at rounds \( r_1 \leq r_2 \), it holds that \( C_1^k \preceq C_2 \). In this context, an honest node is a node that respects the ideal functionality. \( C_1^k \) denotes the chain obtained by pruning \( k \) blocks from \( C \), and \( \preceq \) is the prefix relationship. For transaction confirmation, the common prefix property must hold for 6 blocks \( (C_1^6 \preceq C_2 \text{ for } k = 6) \) [8].

\(^2\)\( M = N \), implies all nodes are the mining nodes (satisfying Nakamoto’s assumption). However, we show in §4, due to centralization, among 36K IP addresses, only 359 belong to mining nodes (\( |M|=359 \) and \( |N|=36K \))
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 $b_r / H$.
- If $P_i \in \mathbb{M}$ 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 $b_{r+1}^\prime \preceq b_r$), $\mathcal{F}$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1} \mathcal{C}_2 \leftarrow b_{r+1}^\prime$. 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 $b_{r+1}^\prime \preceq b_r$), $P_i$ gives time-based precedence to the blocks. i.e., $b_{r+1}$ is received at $t_1$ and $b_{r+1}^\prime$ is received at $t_2$, where $t_2 > t_1$, then $P_i$ only accepts $b_{r+1}$ and discards $b_{r+1}^\prime$ 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 $b_{r+1}^\prime \preceq b_r$), 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 $b_{r+1}^\prime \preceq b_r$), $P_i \notin M$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1} \mathcal{C}_2 \leftarrow b_{r+1}^\prime$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ have an equal length.

onTerminate: On input $(r = r_1)$, $\mathcal{F}$ terminates the execution and proceed towards the evaluation of $Q_{cp}$ and $Q_{cq}$.

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

onValidate: In any round, to validate $Q_{cp}$, $\mathcal{F}$ queries each $P_i \in \mathbb{N}$ to report $\text{VIEW}^P_i$. If $\mathcal{F}$ receives a single ledger $\mathcal{C}$ from all $P_i \in \mathbb{N}$, it considers $Q_{cp}$ 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 $\mathcal{C}_1^{1:k} \preceq C_2$ (i.e., two chains share a common prefix). To evaluate $Q_{cq}$, $\mathcal{F}$ selects the longest chain among $\mathcal{C}_1$ and $\mathcal{C}_2$, and computes the experimental value of $\mu_i$. If $\mu_i - \mu_i' = \epsilon$ (negligible), $\mathcal{F}$ assumes $Q_{cq}$ is preserved. Otherwise, $Q_{cq}$ 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 [25, 14]. We also distinctly characterize the behavior of mining ($P_i \in M$) and non-mining ($P_i \notin M$) nodes when they receive blocks.
**Chain Quality Property.** The chain quality property $Q_{cq}$ with parameters $\mu$ and $l$ specifies that for any honest node $P_i$ with chain $C$, it holds that for any $l$ consecutive blocks of $C$, the ratio of honest blocks is at least $\mu$. $Q_{cq}$ ensures that for a sufficiently large value of $l$, the contribution of $P_i$ in $C$ is proportional to $h_i$. Moreover, $Q_{cq}$ assumes that no $P_i \in M$ acquires more than 50% hash rate [13, 21, 17, 19, 32].

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 $\text{VIEW}^k_{Pi}$ 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 [29]. 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.

**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 [8]. 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 $C_1^6 \preceq C_2$ 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:

$$P(x|\lambda) = \left(\frac{e^{-\lambda}x^x}{x!}\right)^k$$  \hspace{1cm} (1)

Now assume a random variable $X$ which represents an event that $C_1^6 \notin C_2$ 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 (1)) becomes:

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

With $\lambda = 1/600$, $k = 6$, and $x = 2$, $P(X)$ is 0.000001. 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, $h_i$ 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 ($\mu_i$). If a miner faithfully respects the protocol in Figure 1, its probability of contributing 1 block becomes $k \frac{h}{H}$. Plugging in the experimental values, the probability is 0.999 ($\mu_i$). Therefore, $\mu_i - \mu_i$ is 0.001. This is negligible ($\epsilon$) as defined in the ideal world functionality Figure 1.

**Key Takeaways.** The ideal world functionality, in Figure 1, characterizes the modus operandi of Bitcoin. Compared to the prior theoretical models [14, 28], we distinctly define the mining nodes and non-mining
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.

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.

3 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.

3.1 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 [24]) 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 [24]. 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 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 \( \approx 7K \) to \( \approx 9K \) addresses at any time. However, unlike [4, 31], we did not want to rely purely on Bitnodes [7] 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. Second, we used the Bitcoin RPC API for mining nodes detection. In [2], along with the attack countermeasures, we provide an extended version of this paper. In compliance with the submission policy, [2] is kept anonymous. On page 17 [2], we provide more details about challenges in data collection. For measurements, we also sought help from the Bitcoin Core developers to understand the workings of the software.

### 3.2 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 \( \approx 10K \) 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 extended version [2] for details), our crawlers logged the information and took the network snapshot at the time of receiving the block message.
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,
  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.

4 Identifying the Mining Nodes

This section presents the methodology of identifying IP addresses of the mining nodes $M$. The ratio between $|M|$ and $|N|$ exposes the network centralization and can be used to relax the adversarial requirements for partitioning attacks [31].

4.1 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 [9], 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 the extended version [2] page 19). 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 block is added. 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, “lastsrecv” 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 $\text{VIEW}_C$ of a peer $P_i$ with the chain tip at $C$. An update on the tip $C+1$ 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 $\text{VIEW}_C$ to $\text{VIEW}_{C+1}$, if the “synced_headers” value and the inflight value are equal to $C+1$, 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 $N$ with its blockchain tip $C = 570366$ (“synced_blocks”=$570366$). When the crawler receives an update “570367” from $\text{getblockchaininfo}$, it checks $\text{VIEW}_C$ of all its connected peers using $\text{getpeerinfo}$. One information sample of a connected peer is shown in Figure 4. For each peer, the crawler checks if “synced_headers” value is 570367 ($C+1$). When the mining node relays a block, the “inflight” value is also set to the block height ($C+1$). 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.

### 4.2 Revisiting Partitioning Attacks

We revisit the two notable partitioning attacks in the literature [31, 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** [31]. In the temporal partitioning attack, an adversary connects to all reachable IP addresses in $N$ to isolate the vulnerable nodes that have an outdated view. As such, the threat model in [31] 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 $|N|$=36,360 IP addresses, only $|M|$=359 IP addresses belong to the mining nodes. Since, $|M|<<|N|$, 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 [26]. 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, 31]. However, these results are not particularly surprising: the 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.
Figure 5: Distribution of mining nodes across ASes. Compared to the general distribution of Bitcoin full nodes [31, 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.

5 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 [14]. 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 $\text{VIEW}_{C}^{P_{i}}$ for all connected peers in $N$. For a connected node $P_{i}$, if the blockchain tip $C = b_{i}$ (the previous block), then $P_{i}$ is synchronized with an up-to-date blockchain view. If $C < b_{i}$, 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 ($C = b_{i-1}$). 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 [14]. 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.

We report results in Figure 6 as a violin plot [6]. 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%.
Kernel Density Estimation

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.

5.1 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 [11, 31]. In the following, we contrast our findings with prior works and explore factors behind increasing delay.

Decker et al. [11]. 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 [11], there is a natural delay of at least 41.6 seconds ((1020 − 500) × 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. [31]. In 2018, Saad et al. [31] used the Bitnodes [7] 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 [11]. 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 [11], which is consistent with our findings.

5.2 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. [28] 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. [28] 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. In practice, however, when miners (honest or dishonest) receive a block, they immediately start mining the next block [25]. 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.

**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 `ViewP` 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.

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 $\approx 90\%$ 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 $\approx 90\%$ 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 [14] or non-lock-step synchronous [28, 29]. (2) Logically, this behavior suggests that the mining nodes do 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 delay suggest that the Bitcoin mining nodes have a varying network reachability.

### 6 The HashSplit Attack

To understand the security implications of high block propagation delay in an 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 ($Q_{cp}$) and the chain quality ($Q_{cq}$) properties. HashSplit is a lower bound construction that shows: (1) an adversary with an arbitrary hashing power can violate $Q_{cq}$, (2) an adversary with 26% hash rate can violate both $Q_{cp}$ and $Q_{cq}$ 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 ($\S 3$–$\S 5.2$), we note that the asynchronous network creates a natural partitioning among the mining nodes, which expand the strategy space for an adversary to launch various attacks when combined with the Bitcoin mining policies [25, 28, 13, 11, 14]. HashSplit is one such attack that targets the blockchain safety properties.

It is worth noting that, unlike the balance attack [26] or the routing attacks [4, 31], 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 the asynchronous network to split the hash rate. Below, we present the threat model for the HashSplit attack. For more details on the attack novelty compared to other Bitcoin attacks, refer to Appendix $\S A.1$. 
6.1 Threat Model and Attack Objectives

For HashSplit, we assume an adversary $A_m \in M$ with less than 51% hash rate. $A_m$ follows the experiment methodology in §3–§5 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 $P_i \in \mathbb{M}$, $A_m$ maintains a direct connection with them to instantly send or receive blocks. Using the measurement methodology in Figure 7, $A_m$ calculates how a block generated by each $P_i \in \mathbb{M}$ reaches all $P_i \in \mathbb{N}$. If $A_m$ samples the block propagation pattern of each $P_i \in \mathbb{M}$, Figure 7 can be expressed in terms of the general model in Figure 8.

Figure 8 shows that $A_m$ has a good network reachability like Miner A in Figure 7, while $H_m$ (an honest miner) has poor network reachability like Miner B in Figure 7. Since Figure 7 is sampled at one second interval, $A_m$ knows precisely (in seconds) at what time each $P_i \in \mathbb{N}$ receives a block. By calculating the difference in the block generation time and the time at which each $P_i \in \mathbb{N}$ receives the block, $A_m$ can calculate the delay in the block reception for each $P_i \in \mathbb{N}$. For simplicity, and for each $P_i \in \mathbb{M}$, we define the reachability time $T_{i,j} = [t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}]$ as four discrete time indexes at which the block is received by 50% miners, 100% miners, 50% network, 100% network.

We further assume that each $P_i \in \mathbb{M}$, except $A_m$, conforms to the ideal functionality such that when any $P_i \in \mathbb{M}$ generates a block, it immediately releases the block to the network without withholding. Moreover, when a $P_i \in \mathbb{M}$ receives two blocks with a hash pointer to the same parent block, $P_i \in \mathbb{M}$ 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 [25] and is currently deployed in all Bitcoin Core versions. Finally, we assume that (1) $A_m$ cannot influence the communication model of other $P_i \in \mathbb{N}$ through a routing attack [4, 26], and (2) there is no other attack (i.e., selfish mining) taking place concurrent with HashSplit. We specifically model HashSplit for a weaker adversary as a lower bound construction. Logically, the attack is more favorable for a stronger adversary considered in prior works on Bitcoin partitioning attacks [4, 11, 31, 28].

**Attack Objectives.** Given that $A_m$ is a miner with a view of the network’s communication model, $A_m$ can: (1) deviate from the ideal functionality and violate $Q_{cp}$ and $Q_{cq}$, (2) waste the mining power of honest miners, and (3) prevent non-mining nodes from generating or receiving $k$-confirmed transactions [25]. In HashSplit, $A_m$ achieves these goals by exploiting block propagation delay to split the public chain into two branches $C_1$ and $C_2$, and the mining nodes $M$ into two groups $M_1$ and $M_2$. In a perfect split, $A_m$ splits the network hash rate into $C_1 \leftarrow \alpha = 0.51$ and $C_2 \leftarrow \beta = 0.49$ ($\alpha + \beta = 1$), and mines on the branch with a higher hash rate. To violate $Q_{cp}$ for any $P_i \in \mathbb{N}$, $A_m$ ensures that $C_1^{[k]} \notin C_2$ for $k = 6$. To violate $Q_{cq}$, $A_m$ ensures that for any $P_i \in \mathbb{N}$, $\mu_i - \mu_i' \neq \epsilon$ (blockchain has disproportionately high adversarial blocks). We show that the HashSplit adversary meets these objectives with a high probability.

6.2 Attack Procedure

6.2.1 Identifying Vulnerable Nodes

To split the blockchain, $A_m$ first identifies the vulnerable mining nodes with a high block propagation delay by running algorithm 1. In algorithm 1, $T_{a,j}$ and $T_{i,j}$ are reachability times for $A_m$ and other $P_i \in \mathbb{M}$, respectively. $A_m$ initializes four lists (aList...dList) and four variables (aMax...dMax). For each $P_i \in \mathbb{M}$, $A_m$ computes the time windows $\delta_1,...\delta_4$ that represent the difference between the block propagation time of $A_m$ and the target mining node. For intuition, we again refer to Figure 7. Assuming Miner A as $A_m$ and Miner B as $H_m$, 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 $P_i \in \mathbb{M}$ relative to $A_m$’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.

6.2.2 Blockchain Splitting

After discovering the vulnerable nodes, $A_m$ splits the blockchain into two branches, $C_1$ and $C_2$, and miners into two groups, $M_1$ and $M_2$, using algorithm 2. We define the combined hash rate of $M_1$ as $\alpha$ and $M_2$ as $\beta$. Algorithm 2 provides two attack strategies to achieve the split.
Algorithm 1: Identifying Vulnerable Mining Nodes

1. **Input:** $T_{i,j}, T_{a,j}$ (reachability time of the adversary and all other mining nodes)
2. **Initialize:** aList, bList, cList, dList
3. **Initialize:** aMax, bMax, cMax, dMax = 0
4. for $i = 0; i < |M|; i++$ do
5.   $\delta_1 = t_{i,1} - t_{a,1}$, aList $\leftarrow \delta_1$
6.   if $\delta_1 > aMax$ then
7.     aMax = $\delta_1$
8.   $\delta_2 = t_{i,2} - t_{a,2}$, bList $\leftarrow \delta_2$
9.   if $\delta_2 > bMax$ then
10.   bMax = $\delta_2$
11. $\delta_3 = t_{i,3} - t_{a,3}$, cList $\leftarrow \delta_3$
12. if $\delta_3 > cMax$ then
13.   cMax = $\delta_3$
14. $\delta_4 = t_{i,4} - t_{a,4}$, dList $\leftarrow \delta_4$
15. if $\delta_4 > dMax$ then
16.   dMax = $\delta_4$
17. **return:** aList, bList, cList, dList, aMax, bMax, cMax

Algorithm 2: Attack Procedure (Split Ledger)

1. **Input:** $M, A_m$
2. **Case 1:** $A_m$ finds $b_{r+1}$ before any $P_i \in M$
3. **Strategy 1:** $A_m$ waits for another $P_i \in M$ to find $b'_{r+1}$. When $A_m$ receives $b'_{r+1}$ from $P_i \in M$, $A_m$ releases $b_{r+1}$ only to $M_1$ before $M_1$ receive $b'_{r+1}$. $A_m$ 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).
4. **Case 2:** Any $P_i \in M$ finds $b'_{r+1}$ before $A_m$
5. **Strategy 2:** $A_m$ 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 $A_m$ finds $b_{r+1}$ before $t_{i,1}$, it releases $b_{r+1}$ to $M_2$ before $b'_{r+1}$ reaches them.
6. **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 $C$ splits into two branches $C_1$ and $C_2$, and the network hash rate into $\alpha$ and $\beta$. A fork is created.

**Strategy 1.** In this strategy, $A_m$ produces a block $b_{r+1}$ before any $P_i \in M$, and withholds it. $A_m$ waits for another $P_i \in M$ to produce a block $b'_{r+1}$. With the apriori knowledge of $b'_{r+1}$ propagation pattern in the network (algorithm 1), $A_m$ 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 [25]). 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 $C_1 \leftarrow \alpha$ and $C_2 \leftarrow \beta$.

**Strategy 2.** In this strategy, an honest miner $P_i \in M$ produces the block $b'_{r+1}$ before $A_m$. Since $A_m$ knows that $b'_{r+1}$ will take $t_{i,1}$ time to reach $M_1$ (see Figure 8), $A_m$ violates the ideal functionality and keeps mining for $b_{r+1}$ until $t_{i,1}$. If $A_m$ succeeds in mining $b_{r+1}$ by $t_{i,1}$, $A_m$ will release $b_{r+1}$ to the other set of miners ($M_2$) to which $b'_{r+1}$ is yet to reach. As a result, and similar to **Strategy 1**, the blockchain is split into $C_1 \leftarrow \alpha$ and $C_2 \leftarrow \beta$. In summary, **algorithm 2** provides two strategies for the adversary to split the chain into two branches.

**Perfect Split.** As described in §6.1, the perfect split leads to $C_1 \leftarrow \alpha = 0.51$ and $C_2 \leftarrow \beta = 0.49$. If $A_m$, with a hash rate $\alpha_1$, mines on $C_1$, we define the combined hash rate of all miners in $M_1$ as $\alpha = \alpha_1 + \alpha_2$. Since $A_m$ knows the block propagation pattern and the hash rate distribution (Figure 8), $A_m$ can time both strategies in **algorithm 2** to achieve the perfect split such that $\alpha = \alpha_1 + \alpha_2 = 0.51$.

Without losing generality, in the rest of the analysis we assume: (1) $A_m$ achieves perfect split from **algorithm 2**, (2) there are four miners in the network ($A_m, h_1, h_2,$ and $h_3$), (3) $A_m$ and $h_1$ mine on $C_1$ with
\( \alpha_1 = 0.26 \) and \( \alpha_2 = 0.25 \), (4) \( h_2 \) and \( h_3 \) mine on \( C_2 \) with hash rates \( \beta_1 = 0.25 \) and \( \beta_2 = 0.24 \), respectively \( (\beta = \beta_1 + \beta_2 = 0.49) \), and (5) \( A_m \) 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. At \( t_{o,1} \), \( A_m \)'s block reaches \( h_1 \), and reaches both \( h_2 \) and \( h_3 \) at \( t_{o,2} \). Similarly, for \( h_2 \), \( t_{h,1} \) and \( t_{h,2} \) are times at which \( A_m \) and both \( h_2 \) and \( h_3 \) receive a block. We can extend the same propagation sequence for \( h_3 \).

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, our lower bound construction holds in practice.

6.2.3 Block Race

Once the perfect split is achieved, the two chains, \( C_1 \) and \( C_2 \), enter in a block race. To formally analyze the race conditions, we first revisit the mathematical underpinnings of the 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 \( \text{SHA256(SHA256(Header))} < d = \epsilon [0,2^{256} - 1] \). On average, a miner computes \( m = 2^{256}/d \) hashes to mine a block [18]. 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 \( \tau \) [18]. When the hash rate is split into \( \alpha \) and \( \beta \) (algorithm 2), the time required to mine the next block on each branch becomes \( t_o = m/\alpha \) and \( t_o' = m/\beta \). In other words, after executing algorithm 2, the next block from \( C_1 \) is mined at \( t_o \), and at \( t_o' \) for \( C_1 \), respectively. Therefore, the probability that \( C_1 \) succeeds in producing the block before \( C_2 \) becomes \( t_o/(t_o + t_o') = \alpha/(\alpha + \beta) \) [18, 19, 30]. Similarly, the probability that \( A_m \) mines the next block on \( C_1 \) before \( h_1 \) is \( \alpha_1/\alpha_1 + \alpha_2 \), and the probability that \( h_1 \) mines the next block on \( C_1 \) before \( A_m \) is \( \alpha_2/\alpha_1 + \alpha_2 \). This analysis can be easily extended to \( h_1 \) and \( h_2 \) on \( C_2 \).

After executing algorithm 2, \( A_m \) needs to maintain the fork for \( k \) consecutive blocks to violate \( Q_{cp} \). However, if the fork gets resolved and the resulting chain has more blocks than 100\( \alpha_1 \) (i.e., out of 100 blocks, more than 26 mined by \( A_m \)), \( Q_{cq} \) is violated. Note that since there are two public chains, if the fork gets resolved before \( k \), and \( C_1 \) is the winning chain, \( Q_{cq} \) is violated even when \( Q_{cp} \) is preserved. Considering these cases, in the following we concretely specify the race conditions under which the HashSplit attack succeeds or fails:

1. If the forks persist for more than \( k \) blocks, \( Q_{cp} \) is violated, and the attack succeeds partially.
2. If the forks get resolved before \( k \) blocks and \( C_1 \) wins, \( Q_{cq} \) is violated, and the attack succeeds partially.
3. If the forks persist for \( k \) blocks and get resolved at \( k + 1 \) block with \( C_1 \) as the winning branch, both \( Q_{cp} \) and \( Q_{cq} \) are violated, and the attack succeeds completely.
4. If the forks persist for \( k \) blocks and get resolved at \( k + 1 \) block, with all \( k \) blocks mined by \( A_m \), both \( Q_{cp} \) and \( Q_{cq} \) are violated. However, the HashSplit attack becomes a majority attack. In a synchronous network, this probability is 0.08 with \( \alpha_1 = 0.26 \) [30].
5. If the forks get resolved before or after \( k \) blocks and \( C_2 \) wins, \( A_m \) loses all blocks, and the attack fails.

Clearly, HashSplit relies on the block race outcomes in which the forks persist or get resolved. In Figure 9, we formally analyze all outcomes of a block race along with their probability distribution and \( A_m \)'s strategies. We define a random variable \( X \) that specifies the probability distribution of the block race outcome in Figure 9. We further define \( F \) and \( R \) as the sum of events in which forks persist or get resolved. In (3) and (4), we show the probability \( P[X = F] \) and \( P[X = R] \).
Plugging the hash rate of each miner from our threat model, \( P[X = F] \) and \( P[X = R] \) become 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 \( Q_{\text{cp}} \).

2. The probability that a fork is resolved by an honest miner on \( C_1 \) is \( \alpha_2(1 - \alpha_1)(1 - \beta_1)(1 - \beta_2) = 0.1275 \); significantly less than 0.6892 and favors \( Q_{\text{cp}} \)'s violation.

3. The probability that a fork is resolved by any honest miner on \( C_2 \) is \( \beta_1(1 - \alpha_1)(1 - \alpha_2)(1 - \beta_1) = 0.2401 \) (significantly less than 0.6892), which is the failure probability for HashSplit.

4. With \( M \) miners, potentially \( M \) branches can appear after a block race, although with a negligible probability \( \left( \prod_{i=1}^{M} h(i) \right) \). More branches increase the probability of violating \( Q_{\text{cp}} \), and we show in Figure 9 how \( A_m \) can deal 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 \( P[X = F] \), or a single branch with probability \( P[X = R] \) [13, 22]. Figure 10 presents a state machine in which \( S_0 \) and \( S_1 \) denote the states of forks and no forks, respectively. The transition probabilities \( p_{00}, p_{01}, p_{10}, \) and \( p_{11} \) are \( P[X = F], P[X = R], P[X = F], \) and \( P[X = F] \), 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 \( Q_{\text{cp}} \) and \( Q_{\text{cq}} \).

**Incorporating Propagation Delay Advantage.** Before computing the stationary distribution of Figure 10, it is important to incorporate \( A_m \)'s mining advantage due to the block propagation delay and block withholding. For instance, in \( f_1 \), when \( A_m \) produces a block and withholds until \( h_2 \) or \( h_3 \) produce blocks, \( A_m \) 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 additional time for \( A_m \) to mine the next block. To model this advantage, we first 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 \( A_m, h_1, h_2, \) and \( h_3 \) mine blocks with hash rates \( \alpha_1, \alpha_2, \beta_1, \) and \( \beta_2 \), respectively. The relationship between block propagation delay and the hash rate can be obtained as:

\[
\begin{align*}
\alpha_1 &= \frac{\tau}{t_{a,0}}, \quad \alpha_2 = \frac{\tau}{t_{h,1,0}}, \\
\beta_1 &= \frac{\tau}{t_{h,2,0} + t_{h,1}}, \quad \beta_2 = \frac{\tau}{t_{h,3,0} + t_{h,1}}
\end{align*}
\]
Figure 10: State machine representation of a block race. Transition probabilities are $p_{00}$, $p_{01}$, $p_{10}$, and $p_{11}$ are $P[X=F]$, $P[X=\text{R}]$, $P[X=F]$, and $P[X=F]$, respectively.
Considering \( \alpha_1 = 0.26 \), \( \alpha_2 = 0.25 \), \( \beta_1 = 0.25 \), \( \beta_2 = 0.24 \), and \( \tau = 600 \) seconds, from (5), \( t_{a,0}, t_{b,0}, t_{b,0} \) become 2308, 2400, 2400, and 2500, respectively. Plugging these values in (6), 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 \( A_m \)’s advantage in a block race, we convert \( \delta_1 \) in algorithm 1 as the mining advantage that increases \( \alpha_1 \). In our model, \( \delta_1 \) with respect to all honest miners is \( 52 - 2 = 50 \) seconds. In 50 seconds, \( A_m \) gets \( \delta_1/\tau = 50/600 \) fraction of additional mining power over all other miners. As a result, the effective hash rate of each miner becomes \( \alpha_1 = 0.3423 \), \( \alpha_2 = 0.2163 \), \( \beta_1 = 0.2163 \), \( \beta_2 = 0.2073 \). Moreover, \( \mathbb{P}[X = F] \) and \( \mathbb{P}[X = R] \) become 0.739 and 0.261.

This advantage can be extended to miners when they resolve forks \( (r_1 \) and \( r_2 \) in Figure 9). If resolved, 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 \( A_m \), and \( t_{b,1} \) advantage over other miners. Empirically, \( t_{a,1} \) accounts for \( 2/600 = 0.0033 \), and \( t_{b,1} \) accounts for \( 52/600 = 0.087 \) fraction of the mining power. Therefore, if the fork gets resolved, the probability that a fork appears in the next round becomes \( \mathbb{P}[X = F] = 0.683 \). Using these values, we can construct the transition probability matrix for Figure 10.

\[
P = \begin{bmatrix}
S_0 & S_1 \\
S_0 & S_1
\end{bmatrix} = \begin{bmatrix}
\mathbb{P}[0,0] & \mathbb{P}[0,1] \\
\mathbb{P}[1,0] & \mathbb{P}[1,1]
\end{bmatrix} = \begin{bmatrix}
0.739 & 0.261 \\
0.683 & 0.317
\end{bmatrix}
\]

In (7), 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 \( \psi \) such that \( \psi P = \psi \).

\[0.739\psi_1 + 0.261\psi_2 = \psi_1, \quad 0.683\psi_1 + 0.317\psi_2 = \psi_2, \quad \psi_1 + \psi_2 = 1 \]

(7)

From (7), \( \psi_1 = 0.724 \) and \( \psi_2 = 0.276 \), and the long term probability of a forked chain is significantly greater than that of a single branch. Using the stationary distribution, we evaluate the impact of HashSplit on \( Q_{cp}, Q_{cq} \), and the majority attack.

**Common Prefix Property.** Our analysis reveals for any block race of length \( k \), \( Q_{cp} \) is violated \( (c_1 \mid k \notin C_2 \) for any \( k \) with 0.724 probability. For \( k = 6 \), \( P^6 \) yields \( \mathbb{P}[X = F] = 0.72 \). Therefore, HashSplit violates \( Q_{cp} \) with a high probability.

**Chain Quality Property.** Per (6), the block propagation affects the hash rate of each miner. As such, and even when not partitioning the blockchain, \( A_m \) can still mine more blocks that its hash rate allows. For instance, assuming an honest block race, and since \( \delta_1 = 50 \) seconds, \( A_m \) has \( 50/(3 \times 600) \) fraction of mining advantage over the other three miners \( (0.26, 0.223, 0.223, 0.213) \). Moreover, if 100 blocks are mined, \( A_m \) will mine 28.29 blocks. From the ideal-world functionality view, \( \mu - \mu' = 2.29 \neq \epsilon \). \( A_m \) mines two blocks more than its hash rate, thus \( Q_{cq} \) is violated.

**Common Prefix and Chain Quality.** To violate \( Q_{cp} \) and \( Q_{cq} \), a fork needs to persist or get resolved after \( k \) blocks, and \( C_1 \) is the winning branch. Figure 9 shows that \( r_2 \) is the only outcome where forks get resolved to \( C_2 \) with probability 0.2401. We analyze that by branching \( S_1 \) in Figure 10 into two states and calculate the probability of \( C_2 \) being the winning chain (computed as 0.167). Therefore, both \( Q_{cp} \) and \( Q_{cq} \) are violated with a probability of \( 1 - 0.167 = 0.833 \).

**Majority Attack.** From Figure 9, a majority attack happens if (1) \( C_1 \) is the winning branch after \( k \) rounds, and (2) all blocks on \( C_1 \) are mined by \( A_m \). This happens if for \( k - 1 \) rounds, one of the events \( f_i \), for \( i = 1, 2, 3, 4, 7 \) or 9 occurs, followed by event \( f_1 \) 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 \( f_i \), for \( i = 1, 2, 3, 4, 7 \) or 9, while \( S_1 \) determines the probability of \( f_1 \) for \( i = 5, 6, \) or 8, and \( r_1 \) or \( r_2 \). From Figure 9, we compute \( \mathbb{P}[0,0], p_{01}, \) \( p_{10} \), and \( p_{11} \) as 0.663, 0.337, 0.576, and 0.424, respectively. For \( k = 6 \), the result is \( (0.63 \times 0.342) = 0.2156 \). Therefore, with a probability of \( 0.2156 \), HashSplit allows \( A_m \) to launch a majority attack with only 26% hash rate. In the lock-step synchronous or non-lock-step synchronous networks, the probability of successful majority attack with 26% hash rate is \( \approx 0.08 \) [30].

In summary, HashSplit violates the blockchain safety properties with a high probability and significantly amortizes the cost for the majority attack. In this paper, we have only presented an attack against the mining nodes, although it can be launched against non-mining nodes (i.e., Bitcoin exchanges) to prevent them from
generating $k$–confirmed transactions. As shown in §5, the non-mining nodes have a relatively poor network synchronization compared to the mining nodes, making them more vulnerable to HashSplit. As noted in §1, splitting the mining power to amortize the cost of the 51% attack is known in the literature [26, 16, 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 communication, and only relies on the latency and mining policies to split the network. In the past 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 network deployment.

We show the attack feasibility by discrete event simulations, and the results are in Appendix A.2.

7 Attack Countermeasures

In this section, we instead focus on attack countermeasures. Since HashSplit primarily exploits asynchronous network and block propagation delay, if $\delta_1 \ldots \delta_4$ in algorithm 1 are minimized, $A_m$: (1) cannot split the 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 exhibit a lock-step or non-lock-step synchronous network, countering HashSplit.

To reduce propagation delay and form $M \times M$ network topology, we modified and released a Bitcoin Core version to counter the attack (anonymously released [2]). In our Bitcoin Core version, we modified the source code to allow fast connectivity with Bitcoin nodes. Existing Bitcoin Core suffers from poor network reachability, and a default node takes $\approx 120$ days to connect to only 125 IP addresses (out of 6K–10K) [33], a long duration for a small fraction of the total network. To overcome this shortcoming, we modified added scripts to enable faster connectivity, as outlined in [2].

For performance evaluation, we deployed our Bitcoin Core client on a local machine and evaluated the connectivity speedup and bandwidth consumption, with results reported in Figure 11. Our node connected with over 6K reachable nodes in less than 100 seconds, with a bandwidth consumption under 6Mbps (4Mbps incoming and 2Mbps outgoing) during the initial connectivity phase. Once the number of connections stabilize, the bandwidth consumption becomes $\approx 4$Mbps. Our Bitcoin Core version is still in the testing phase and currently supports fast connections to nodes running IPv4 or IPv6.

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 $A_m$’s advantage. However, we acknowledge that $M \times M$ topology does not fully counter the attack. Our measurements show that the network latency can be heterogeneous 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). Heterogeneous latency can be leveraged by $A_m$ 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 fork resolution mechanism. We note from Figure 9, that the victim nodes have multiple branches of the same length in each round (i.e., $C_1$ and $C_2$) during the attack. Particularly, miners on $C_1$ will continuously receive blocks from $A_m$, immediately
followed by blocks from other honest miners. We leverage this sequence of block arrival to eliminate $A_m$’s advantage and reduce the likelihood of a perfect split. In our Bitcoin Core [2], we provide a fork resolution mechanism in which a node removes the connection and bans the IP address for twenty four hours in the event of receiving $k = 6$ sequential blocks from it. This means that $A_m$ will lose a direct connection to all mining nodes and will not be able to achieve a perfect split. $A_m$ may deploy Sybil nodes in the network to connect to the victim. However, in that case $A_m$ will lose $\delta_1$ 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 fork resolution mechanism can mitigate the HashSplit attack.

8 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 of 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 only validate the asynchronous execution of Bitcoin which helped us to construct the threat model and the attack procedure. The second, we ignored the churn among the non-mining nodes, as we note 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 $\approx 10K$ IP addresses were reachable at any given moment, including $359$ addresses of mining nodes. The non-mining nodes often changed their IP addresses resulting in churn. We noticed that churn occurs due to cloud hosting, where 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, churn does not affect our results. Moreover, our proposed countermeasures provide means to mitigate the attack by creating a lock-step synchronous network.

**Conclusion.** In this paper, we formulate the Bitcoin ideal functionality, identify the mining nodes, demonstrate weak network synchronization, and show the network asynchrony in the real world. Across various measures, we show that the Bitcoin network is evolving, where known attacks can be optimized and new attacks can be launched, as demonstrated by HashSplit. Our work bridges the gap between theory and practice of blockchain security, validated by large-scale measurements, and drawing attention to Bitcoin vulnerabilities.

References


In the following, we provide the proof for the ideal world functionality (§2 and Figure 1).

A Algorithmic Analysis of Bitcoin Consensus

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. [14] 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. [28] that analyzes the performance of Blockchain protocols in asynchronous networks.3

The Bitcoin Backbone Protocol [14]. 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 [29]. Moreover, the model assumes $M = N$, where the hash rate is uniformly distributed among all miners. Finally, the adversary does not control more than $|M|/2 = |N|/2$ miners. In other words, the miner is bounded by 50% hash rate. Using these assumptions, the [14] 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, $f$ 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 $l \geq 2\lambda f$.

---

3Although Pass et al. call their model asynchronous, however, Ren et al. [29] show that their model is actually non-lock-step synchronous. For simplicity, in this section, we still refer to [28]'s model as asynchronous.
Although the Bitcoin backbone protocol in [14] 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 $M = N$ and the mining power to be uniformly distributed. However, as shown in §4, $M << N$ 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 [14], this means that if the adversary controls $\approx |M|/4$ or $|N|/4$ miners, the common prefix property and the chain quality property will not hold.

**Blockchains in non-lock-step Synchronous Networks [28].** Improving the model of Garay et al. [14], Pass et al. [28] proposed an non-lock-step synchronous model to evaluate Bitcoin. Their proposition introduces a network delay parameter $\Delta$ that an adversary can add during block propagation. By the time the block reaches other miners, the adversary leverages $\Delta$ to gain a head start towards computing the next block. In the following, we analyze the model proposed in [28].

(1) The primary assumption of their model is that an adversary is able to compute a block, delay its transmission by an upper bound $\Delta$, 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 [14]. Therefore the key difference in the non-lock-step synchronous model [28] and the lock-step synchronous model [14] is $\Delta$, after which both models emulate the same behavior. (2) $\Delta$ gives an advantage to the adversary over all the other participants. Roughly speaking, in the $\Delta$ time window, the adversary is able to mine on top of its previous block. Moreover, [28] assumes that during $\Delta$, 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 $\rho$ 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 $\Delta$ is plugged into the system, the model in [28] 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 $p < \frac{1}{\rho m\kappa}$, consistency is satisfied. (3) The bounded delay $\Delta$ also “discounts” the capability $\alpha$ of honest nodes to produce blocks in a round. To formally capture that, the model in [28] introduces $\gamma = \frac{\alpha}{1 + \Delta \alpha}$, which is the discounted version of $\alpha$ or the effective mining power gained by delaying the block propagation by $\Delta$.

Using the aforementioned assumptions and a security parameter $\kappa$, [28] proposes two theorems to characterize Nakamoto consensus specific to the Bitcoin operations.

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

$$\mu_{\text{nak}}^p(\kappa, n, \rho, \Delta) = 1 - (1 + \delta)\frac{\beta}{\gamma}$$

(8)

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

$$\alpha(1 - (2\Delta + 2)\alpha) \geq (1 + \delta)\beta$$

(9)

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

The consistency property in (9) can also be interpreted as the common prefix property in [14]. The $T$-consistency specifies that the two ledgers must share a common prefix after pruning $T$ blocks from their chains ($C_1^T \preceq C_2$). Moreover, in [28] (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 ($C_1^{T_k} \preceq C_2$, 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 [28], where an non-lock-step synchronous model starts emulating the lock-step synchronous model.

**Experimental Interpretation.** The experimental setup in [28] 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.

### A.0.1 Non-lock-step Synchronous Model in Real World

Although the non-lock-step synchronous model in [28] 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. Firstly, [28] assumes that during $\Delta$, other mining nodes remain idle, and after $\Delta$, the system abruptly starts emulating lock-step synchronous behavior. This would imply that when an adversary delays a block by $\Delta$, no other miner receives the block in the meantime, and after $\Delta$, 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 $\Delta_1$, $P_2 \in M$ receives the block after $\Delta_2$, and $P_w \in M$ receives the block after $\Delta_w$. 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 [28]. They will be mining on top of the block $b_r$. Therefore values of $\alpha, \gamma$, and $\delta$ in Theorem 4, Theorem 5 will change. In the case that we have outlined above, $\alpha$ will become $\alpha_1,...,\alpha_w$, and similarly $\delta$ and $\gamma$ would change to $\delta_1,...,\delta_w$ and $\gamma_1,...,\gamma_w$, respectively. The value of $\beta$ will however remain unchanged. As a result, in the real world settings, the overall advantage of the adversary, due to $\Delta$ will decrease. Plugging this into [28], 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. [11]. 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.

### A.1 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 [13, 12]. 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 its success probability as \( a_{s+1} \). Conversely, if the attacker finds the next block with probability, the gap will reduce by \( s - 1 \), with its success probability as \( a_{s-1} \). Given this information, \( a_s \) must satisfy the following recurrence relationship.

\[
\begin{align*}
a_s &= \frac{(H - h_a)a_{s-1} + (h_a)a_{s+1}}{H - h_a} \\
    &= \min\left(\frac{h_a}{H - h_a}, 1\right)^{\max(s+1,0)} \\
a_s &= \begin{cases} 
1 & \text{if } s < 0 \text{ or } h_a > (H - h_a) \\
\frac{h_a}{H - h_a}^{s+1} & \text{if } s > 0 \text{ or } h_a < (H - h_a)
\end{cases}
\end{align*}
\]  

(10) \hspace{1cm} (11) \hspace{1cm} (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 [30]. Instead, it keeps on extending its private chain in hopes of attaining 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 [20]. 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
Figure 12: 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 $t_{n,2}$. Adversary releases the chain after 8th block.

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.

A.2 Simulations and Results

In the following, we demonstrate HashSplit through computer simulations. We developed a simulator in Python that implements the PoW protocol. 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 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 future, therefore, it can be tailored to any custom topology implementation.

Figure 12 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 chain quality.