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

Abstract

The Bitcoin blockchain *safety* relies on strong network synchrony. Therefore, violating the blockchain *safety* requires strong adversaries that control a mining pool with ≈51% hash rate. In this paper, we show that the network synchrony does not hold in the real world Bitcoin network which can be exploited to lower the cost of various attacks that violate the blockchain *safety* and chain quality. Towards that, first we construct the Bitcoin ideal 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 ideal functionality against the real world measurements to expose the network anomalies that can be exploited to optimize the existing attacks. Particularly, we observe a non-uniform block propagation pattern among the mining nodes showing that the Bitcoin network is asynchronous in practice.

To realize the threat of an 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 tweaking Bitcoin Core to model the Bitcoin ideal functionality. Our measurements, theoretical modeling, proposed attack, and countermeasures open new directions in the security evaluation of blockchain systems.

1 Introduction and Related Work

Bitcoin is a dynamically evolving distributed system that has significantly scaled up in recent years [9]. As Bitcoin continues to grow and inspire other decentralized applications, its security features are continuously investigated using theoretical analysis and measurement techniques [25, 31]. 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 [29, 13, 32, 34] 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 [3, 26].[1] 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.* [13], 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 [10]. To address this limitation, Pass *et al.* [32] extended the work in [13] and analyzed Bitcoin in the *non-lock-step* synchronous settings [34]. 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 6), which cannot be modeled as a uniform advantage. In §A, we analyze the shortcomings of these models along with refinements to their assumptions. In §5.1, 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 blockchain *safety* properties, namely the common prefix and the chain quality [13]. In §6, we show that the asynchronous network relaxes the requirement to violate these two properties.

Measurement Studies. In addition to the theoretical models, notable works on network measurements have focused on (1) analyzing Bitcoin nodes distribution across autonomous systems (ASes) [3, 36, 14],

[1] The Bitcoin network consists of full nodes and SPV clients. Among the full nodes, there are mining and non-mining nodes. The mining nodes are used by the mining pools to broadcast blocks in the network. In [26, 3], 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.

(2) discovering influential nodes controlled by the mining pools [26, 3], and (3) measuring the network synchronization [10, 36]. The security evaluations of these studies proposed (1) partitioning attacks through BGP prefix hijacking of high profile ASes [3], (2) majority attacks with less than 51% hash rate (≈49% in [10]), and (3) a combination of the two attacks (*i.e.,* spatio-temporal partitioning in [36] and the balance attack in [30]). In the context of measuring the network synchronization, the two related studies to our work are [10] and [36].

In 2012, Decker *et al.* [10] conducted the first measurement study to analyze the Bitcoin network synchronization. They concluded that the block size is the dominant factor in blockchain synchronization. In their measurements, they connected to ≈3.5K IP addresses and observed that ≈90% of the nodes in the network receive the newly published block within 12.6 seconds on average.[2] In contrast, our measurements reveal a relatively weaker synchronization where a large number of the connected nodes do not synchronize on a newly published block. The observed differences are likely due to (1) an increase in the network size from 16K reachable IP addresses in 2012 [10] to ≈36K reachable IP addresses at the time of conducting this study, (2) an increase in the block size from 500KB to 1MB, and (3) an increase in the number of low bandwidth nodes and Tor nodes. In 2019, weak network synchronization was also reported in [36] using Bitnodes' dataset. The authors observed a few instances reported by Bitnodes where a majority of the network nodes did not synchronize on the blockchain.

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 [13, 32] and ignore the critical distinction between the mining and non-mining nodes (*i.e.,* in [36, 10, 30]). 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 patterns among the mining nodes which exposes the asynchronous network. Therefore, these studies have assumed a synchronous network which only allows limited attack strategies [32]. 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 measurements and their deviation from the ideal functionality, requirements for existing attacks can be 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 [3], the authors show that BGP attacks can reduce the mining power of the Bitcoin network. In [30], Natoli *et al.* used the routing attack model to present a trade-off between the network delay and the adversary's mining power (also simulated by Gervais *et al.* [15]). Similarly, the Eclipse attack [19] showed that an adversary can influence the hash rate of the mining nodes by occupying all their incoming and outgoing connections. However, all these attacks rely on disrupting the network communication to create a split between the mining nodes. Therefore, they implicitly assume a form of route manipulation (*i.e.,* BGP hijacking or occupying incoming and outgoing connections [30, 19, **?**]) as a prerequisite to introduce delay and split the mining power. In contrast, we show that the non-uniform delay in the existing block propagation patterns can be exploited to split the mining power without disrupting the communication model through route manipulation or connection control. We show that by only leveraging the observed block propagation pattern among the mining nodes and selective block broadcast, an adversary can violate the *safety* properties of the Bitcoin blockchain.

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

1. We construct the Bitcoin ideal world functionality to formally specify the *safety* properties of the Bitcoin ledger; the common prefix property and the chain quality property [13] (§2). The ideal world functionality faithfully models the expected functionality of a correct Bitcoin implementation across prevalent

[2] Another interpretation of this result is that 12.6 seconds after the release of a new block, 90% of the nodes synchronized on that block and added it to their blockchain.

deployments in real world Bitcoin network.

2. We deploy crawlers in the Bitcoin network and connect with over 36K IP addresses in five weeks (§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 during the average inter-arrival block time, a large number of connected nodes do not synchronize on the blockchain. Moreover, through a fine-grained analysis of the block propagation patterns, we show that execution of the Nakamoto consensus in Bitcoin is asynchronous §5.1.

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 high probability. We also propose attack countermeasures by tweaking Bitcoin Core in order to closely model the ideal functionality [1].

The *HashSplit* attack is a lower bound construction and it can be launched as long as there is a non-uniform block propagation in the network. The attack is based on the gaps 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 measurements and the attack. Moreover, the paper includes discussions and conclusions in §8, and appendices with supplementary findings in §A–§G.

2 The Bitcoin Ideal Functionality

In this section, we present the Bitcoin ideal world functionality, which we later contrast with our measurements to present the *HashSplit* attack. The Bitcoin white paper assumed a network where each node possessed the capability of solving PoW (1 CPU=1 Vote) [29]. However, over time, the PoW difficulty has significantly increased, allowing only a few nodes to solve it. This change occurred due to large mining pools that create the mining centralization [39].

Acknowledging 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 [13, 32], 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 [13]. In Appendix §B, we explain the model assumptions and the main theorems derived in [13, 32] by presenting the experimental interpretation of their results in the context of our ideal functionality.

First, we define $\mathbb{N}$ as the set of all reachable IP addresses of Bitcoin nodes. We define $\textsc{View}_{\mathcal{C}}^{P_i}$ as the blockchain view of a single node $P_i \in \mathbb{N}$, where $\mathcal{C}$ is the blockchain ledger. The Bitcoin backbone protocol [13] states that the inter-arrival time between two blocks must be sufficiently large that each $P_i \in \mathbb{N}$ has $\textsc{View}_{\mathcal{C}}^{P_i}$ (*i.e.,* in ≈10 minutes, all $P_i \in \mathbb{N}$ have the up-to-date blockchain). Next, we define $\{M \subset \mathbb{N}\}$ as a set IP addresses of the mining nodes.[3] For each $P_i \in M$, h_i is P_i's hash power, where $0 < h_i < 1$. $H = \sum_i^{|M|} h_i = 1$ is the total hash power of all the mining nodes. With the network entities defined, below, we discuss the common prefix property and the chain quality property of the Bitcoin blockchain.

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

Chain Quality Property. The chain quality property $\mathcal{Q}_{cq}$ with parameters μ and l specifies that for any honest node P_i with chain $\mathcal{C}$, it holds that for any l consecutive blocks of $\mathcal{C}$, the ratio of honest blocks is at least μ. $\mathcal{Q}_{cq}$ ensures that for a sufficiently large value of l, the contribution of P_i in $\mathcal{C}$ is proportional to its

Ideal World Functionality of Bitcoin

Input: Nodes $\mathbb{N}$ including miners M, blockchain $\mathcal{C}$, and trusted party $\mathcal{F}$. The protocol starts at round $r = r_0$ for a length l. Prior to the execution, each $P_i \in M$ reports its hash rate h_i to $\mathcal{F}$, using which $\mathcal{F}$ computes $\mu_i^{'}$, the expected chain quality parameter for each P_i. $\mathcal{F}$ mandates that $h_i < 0.5H$, $\forall P_i \in M$; otherwise, $\mathcal{F}$ aborts. 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 fully connected, allowing each P_i to communicate with any node in $\mathbb{N}$ or $\mathcal{F}$, concurrently.

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

- Each round r, each $P_i \in M$ computes b_{r+1} with probability $\frac{h_i}{H}$.
- If $P_i \in M$ finds b_{r+1} before it receives b_{r+1} from any other miner, it broadcasts b_{r+1} to $\mathcal{F}$ and $\mathbb{N}$ (no block 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 round r 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 $P_j \in M$ in round r, then P_i stops its own computation for b_{r+1}, extends the chain $\mathcal{C} \leftarrow b_{r+1}$, and moves to the next round to compute the next block using b_{r+1} as the parent block.
- If $\mathcal{F}$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), $\mathcal{F}$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1}$ $\mathcal{C}_2 \leftarrow b_{r+1}^{'}$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ have an equal length.
- If $P_i \in M$ receives multiple inputs for the same parent block (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), P_i gives time-based precedence to the blocks. *i.e.,* b_{r+1} is received at t_1 and $b_{r+1}^{'}$ is received at t_2, where $t_2 > t_1$, then P_i only accepts b_{r+1} and discards $b_{r+1}^{'}$ by treating it as an orphaned block. P_i extends the chain $\mathcal{C} \leftarrow b_{r+1}$ and moves to the next round to compute the next block using b_{r+1} as the parent block.
- If $P_i \in M$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), at the same time t_1, P_i tosses a coin and selects one of the two blocks to extend the chain.
- If $P_i \notin M$ receives multiple inputs for the same parent block in a round (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), $P_i \notin M$ forms two concurrent chains $\mathcal{C}_1 \leftarrow b_{r+1}$ $\mathcal{C}_2 \leftarrow b_{r+1}^{'}$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ have an equal length.

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

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

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

Figure 1: The Bitcoin ideal functionality closely modeled on the practical implementation of Bitcoin as we largely see it. We use P_i to denote any node in the network. If P_i is among the mining nodes $P_i \in M$, then it possesses the hashing power to mine blocks. If P_i is not among the mining nodes $P_i \notin M$, then it simply maintains a blockchain and contributes to network synchronization by relaying blocks to other nodes. The mining nodes M follow the communication model specified in [29, 13].

Model	Network	Topology	Mining Nodes
Garay *et al.* [13]	*lock-step* synchronous	Strongly Connected	✗
Pass *et al.* [32]	*non-lock-step* synchronous	Strongly Connected	✗
This Work	*lock-step* synchronous	Strongly Connected	✓

Table 1: Contrasting our ideal functionality against the prior theoretical models. The key difference is the distinction we make between the mining and non-mining nodes by embracing the mining centrality in the current Bitcoin network.

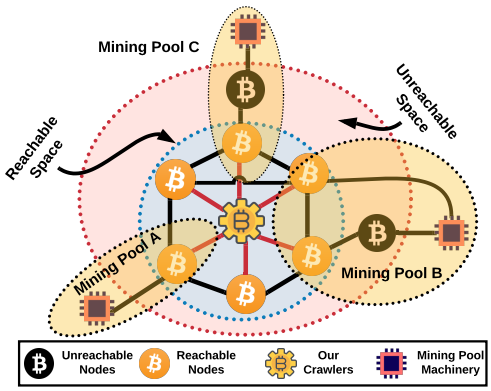


Figure 2: An illustration of our data collection system contextualized in the Bitcoin network. Mining pools can have *reachable* (Mining Pool A), *unreachable* (Mining Pool C), or both (Mining Pool B) types of nodes. Note that *unreachable* nodes cannot connect with each other. Therefore, a block must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

hash rate h_i. Moreover, $\mathcal{Q}_{cq}$ assumes that no $P_i \in M$ acquires more than 50% hash rate [12, 23, 16, 18, 37].

Using these properties, we define the Bitcoin ideal world functionality in Figure 1. Our formulation assumes $P_i \in \mathbb{N}$ as "interactive Turing machines" (ITM) that execute the Nakamoto consensus for l rounds, arbitrated by a trusted party $\mathcal{F}$. A round is a time in which each $P_i \in M$ is mining on the same block. For any $P_i \in M$, a round terminates when the $\text{VIEW}_{\mathcal{C}}^{P_i}$ is updated with a new block. The network $\mathbb{N} \times \mathbb{N}$ is a fully connected such that when a block is released by any $P_i \in M$ at t_1, all nodes receive it at the next time step t_2. As a result, the network exhibits a *lock-step* synchronous execution [34]. Due to varying roles in the system, the mining nodes $P_i \in M$ and the non-mining nodes $P_i \notin M$ have unique operations. For instance, when a $P_i \in M$ receives two valid blocks for the same parent block, it gives time-based precedence to the block received earlier. The block received later is discarded. However, when a $P_i \notin M$ receives two valid blocks, it creates two concurrent branches of the chain and waits for the next block to extend one of them. The ideal world functionality in Figure 1 is consistent with the rules encoded in the current Bitcoin Core version. In Table 1, we contrast our ideal world functionality against the prior theoretical models and in §A, we provide the ideal world functionality proof.

Key Takeaways. The ideal world functionality, in Figure 1, characterizes the *modus operandi* of the Bitcoin network. Compared to the prior theoretical models [13, 32], we distinctly define the mining nodes and non-mining nodes and characterize their unique roles in the system. In the rest of the paper, we perform a data-driven study to investigate (1) the size $|M|$ of the mining nodes, (2) the synchronization patterns in the network to understand how closely Bitcoin follows the ideal functionality, and (3) show gaps between the ideal functionality and measurements to construct *HashSplit*.

3 Data Collection

In this section, we present our data collection system used for conducting measurements and analysis. Prior to highlighting the system details, it is important to discuss the Bitcoin network anatomy and the characteristics of *reachable* and *unreachable* nodes.

3.1 Bitcoin Peer-to-Peer Network

Broadly, there are two types of Bitcoin full nodes, namely the *reachable* nodes and the *unreachable* nodes. The *reachable* nodes establish outgoing connections as well as accept incoming connections from other *reachable* and *unreachable* nodes. The *unreachable* nodes (often behind a NATs [26]) only establish outgoing connections. For simplicity, we can characterize the Bitcoin network between the *reachable* space and the *unreachable* space, as shown in Figure 2.

[3] $M = \mathbb{N}$, implies all nodes are the mining nodes (satisfying Nakamoto's assumption). However, in §4, we show that due to mining centralization, there are only 359 miming nodes among 36K IP addresses ($|M|$=359 and $|\mathbb{N}|$=36K).

It is argued that mining pools prefer to host their mining nodes in the *unreachable* space due to security concerns [26]. As such, if we assume that *all* mining nodes exist in the *unreachable* space, it implies that mining nodes cannot accept incoming connections from other mining nodes, and their blocks will have to be relayed by the 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 and therefore vulnerable to the attack construction presented in §6. Moreover, hosting only the *unreachable* nodes also adds delay in block propagation since the block is first relayed to a *reachable* node which then relays the block to its connections. This delay is undesirable for the miner and the Bitcoin network at large [10]. To further understand these arguments, we reached out to developers and authors of prior works. From our 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 typically host both *reachable* and *unreachable* nodes. (2) Since two *unreachable* nodes cannot directly connect to each other, blocks between the *unreachable* nodes are relayed by the *reachable* nodes. (3) *Reachable* nodes are responsible for relaying blocks and maintaining the network synchronization. (4) This block relaying method is followed even when miners use fast relay networks [28]. (5) Since the *reachable* nodes are the entry points for a block in the *reachable* network (Figure 2), we can mark those entry points and treat them as the mining nodes by connecting to all the *reachable* nodes.[4] (6) The frequency of relaying blocks can be used to estimate the hashing power of the mining pool behind a *reachable* node [26].

Using these insights, we set up a data collection system to connect with the *reachable* nodes. Based on the prior works, we noticed that the number of *reachable* nodes in Bitcoin can vary between ≈6K to ≈9K addresses at any time. However, unlike [36], we did not want to rely solely on Bitnodes [7] for data collection since Bitnodes does not disclose the mining nodes. Instead, we developed our own data collection system and customized it to our desired measurement specifications.

Key Challenges. In 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 (115 incoming and 10 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 those artifacts, 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 parsed the *peers.dat* file to obtain those addresses. Second, we used the Bitcoin RPC API for mining nodes detection. For measurements, we also sought help from the Bitcoin 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* that (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.[5] Figure 2 provides an illustration of our data collection system in the context of Bitcoin peer-to-peer network. We connected to the *reachable* nodes in the network and whenever a mining pool released a block in the *reachable* space, our crawlers marked that node and measured the

[4] Bitcoin network cannot synchronize without *reachable* nodes participating in the block propagation. Therefore, our technique of identifying the mining nodes through the *reachable* nodes is valid even if miners use fast relay network or exchange blocks through non-Bitcoin communication channels.

[5] We found that at any time, there were up to ≈10K *reachable* nodes. Therefore, the difference between the total number of nodes we connected (≈36K) and the maximum number of nodes *reachable* at any time (≈10K) indicates a churn which was predominantly observed among the non-mining nodes.

```
{ 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 3: A sample JSON output when a block is received by our crawler from a 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 relaying to the crawler.

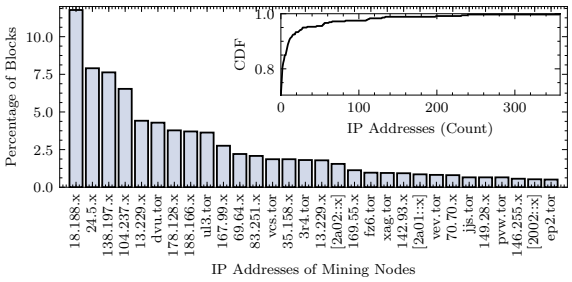


Figure 4: Results obtained by applying Heuristic 1 on our dataset. The histograms show the percentage of blocks contributed by mining nodes. We mask the last two octets to preserve anonymity. The subplot is the CDF showing the distribution of IP addresses with respect to the blocks produced.

network synchronization. Supplementary details on data collection are provided in §C.

4 Identifying the Mining Nodes

Prior works have used block INV messages to detect mining nodes [3].[6] For our experiments, 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.

The Bitcoin RPC API command *getblockchaininfo* provides information about the latest block on the blockchain tip. 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. A sample interaction with one peer is shown in Figure 3 and the key variables to note are "addr", "lastsrecv", "synced_headers", "synced_blocks", and "inflight." "addr" is the connected peer's IP address, "lastrecv" is the latest UNIX timestamp at which the peer relayed any information, "synced_headers" is the last block header message sent by the peer, "synced_blocks" is the last block INV message sent by the peer, and "inflight" is the block relayed by the peer. Viewed through the lens of our ideal world functionality (Figure 1), "synced_blocks" renders the view $\textsc{View}_{\mathcal{C}}^{P_i}$ of a peer P_i with the chain tip at $\mathcal{C}$. An update on the tip $\mathcal{C}+1$ is captured by "synced_headers". Using this information, we developed **Heuristic 1** to detect the mining nodes.

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

[6] An INV message is an announcement of a new block or a transaction by a node to all its connected nodes. In response, the connected nodes may send a GETDATA message to request for the transaction or the block, corresponding to the INV message. The CMPCTBLOCK method is a recent technique [8] that reduces delay incurred in the INV and GETDATA exchange (see Figure 11 for details).

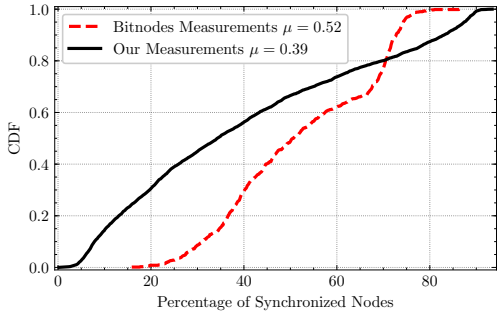


Figure 5: CDF of synchronized nodes reported from our measurements and Bitnodes dataset. Our results show that in ≈9.98 minutes on average, only 39.43% of the connected nodes synchronized on the blockchain. Bitnodes dataset shows that on average, only 52.24% nodes had an up-to-date blockchain at any time. In both cases, the results are different from the expected ideal functionality.

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 3 that shows a sample interaction of a crawler connected to peers in $\mathbb{N}$ with its blockchain tip $\mathcal{C} = 570366$ ("synced_blocks"=570366). When the crawler receives an update "570367" from *getblockchaininfo*, it checks the information of all its connected peers using *getpeerinfo*. One information sample of a connected peer is shown in Figure 3. For each peer, the crawler checks if "synced_headers" value is 570367 ($\mathcal{C}+1$). When the mining node relays a block, the "inflight" value is also set to $\mathcal{C}+1$. The example in Figure 3 shows that the "inflight" value is "570367," hence the "addr" is the node's IP address.

Results. We applied **Heuristic 1** to detect the mining nodes. A summary of results is reported in Figure 4. 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 [2]. Among the 359 mining nodes, 31 nodes produced 80% blocks, and 67 nodes produced 90% blocks.

As discussed in §2, due to the mining centralization, there are a few mining pools that mine the Bitcoin blockchain. Therefore, the number of full nodes hosted by those mining pools ($|M|$) is less than the total number of full nodes ($|\mathbb{N}|$) in the network. In 2014, Miller *et al.* [26] conducted a measurement study, reporting ≈100 mining nodes controlled by 8 mining pools. Simply extrapolating the measurements in [26] to the number of dominant mining pools at the time of conducting our study (25 [39, 33]), an estimation for $|M|$ would be 313 (=100×25/8). As such, the number of mining nodes discovered in our experiments (*i.e.,* $|M|$= 359) is close to the expected number through extrapolation (*i.e.,* 313).[7]

5 Network Synchronization

After identifying the mining nodes, we analyzed the network communication model to validate its compliance with the ideal world functionality. Our main objective was to understand if the block propagation pattern of the mining nodes varies based on their network reachability. As such, variance in the block propagation pattern deviates from the communication model specified in the ideal functionality (Figure 1), which can be exploited to curate new attack strategies. In this section, we first report a high level overview of network synchronization observed in our experiments, followed by a demonstrative example that shows that block propagation pattern varies in the real world Bitcoin network.

For a high level overview of network synchronization, we analyzed the interaction between our crawlers and the nodes to which they were connected. To preserve the common prefix property, the inter-arrival time between two blocks must be long enough to allow all nodes to synchronize on the blockchain [13]. In terms of the ideal functionality (Figure 1), when a new block b_{i+1} is released, if any node $P_i \in \mathbb{N}$ queries its connected peers, it must observe the previous block (b_i) on the blockchain tip of all connected peers. To evaluate this synchronization property, we use **Heuristic 2** below.

[7] Mining nodes detection not only contributes to the *HashSplit* attack but also lowers the cost of launching other partitioning attacks [3, 36], which we discuss in §D.

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

To elaborate on **Heuristic 2**, we again refer to Figure 3. When our crawler received the new block 570367 (b_{i+1}) from the mining node, the "synced_blocks" value was 570366 (b_i). This means that before sending the new block, the mining node's blockchain tip was 570366 ($\mathcal{C} = b_i$). Hence, the mining node had synchronized with 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 strong synchronization. If the value of "synced_blocks" was found to be less than 570366 (*i.e.,* 570365), the peer was noted to exhibit weak synchronization. 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, for all the connected nodes, we were able to observe the percentage of nodes that synchronized with an up-to-date blockchain.

We plot the CDF of synchronized nodes in Figure 5, showing a weak network synchronization. We observed that the average block time was ≈9.98 minutes during which only 39.43% nodes had the synchornized blockchain. Moreover, compared to the measurements in prior works [10, 36] (details in §E), our results indicate that the network synchronization has changed with time.

Result Validation. Given a surprisingly weak network synchronization observed at our crawlers, we decided to validate our findings by crawling recent data from Bitnodes (October 2020 to December 2020). In Figure 5, we plot results from the Bitnodes dataset which also highlights the issues with network (on average, 52.2% of the nodes have a synchronized blockchain at any time). In §E, we further elaborate on the aspects of weak synchronization. It is reasonable to assume that the weak synchronization could increase the number of blockchain forks. Unsurprisingly, 34 forks have been observed on the blockchain since January 2020.[8] [9]

5.1 Bitcoin Network Asynchrony

Exploiting the block propagation pattern to violate the blockchain consistency has been extensively studied in prior works [10, 40, 22]. The most commonly referenced theoretical model in this context is by Pass *et al.* [32] in which they analyzed the Bitcoin blockchain consistency in the *non-lock-step* synchronous network. The *non-lock-step* synchronous network allows an adversary to delay the block by a parameter Δ, giving the adversary a head start mining advantage. Pass *et al.* [32] 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. Although, Pass *et al.* [32] called their model "asynchronous," however, Ren *et al.* [34] showed that the model is actually *non-lock-step* synchronous.

Ren *et al.* [34] further specified that an asynchronous model is weaker than the *non-lock-step* synchronous model, allowing an adversary to maintain a public fork for k blocks. In light of our ideal world functionality, a prerequisite to that model would be that the mining nodes (1) receive blocks at different times, and (2) do not form $M \times M$ topology. In this section, we show that these two observations can be made in the current Bitcoin network, thus satisfying the notion of "asynchrony" established in [34]. In §6, we will further demonstrate how the adversary exploits the asynchronous network to maintain a fork by orchestrating concurrent mining on two branches of the public chain.

Block Propagation Among Mining Nodes. To validate that mining nodes receive blocks at different times and do not form $M \times M$ topology, we conducted a short experiment to precisely study the block propagation among the mining nodes. For this experiment, we executed the *getpeerinfo* command at one second interval, allowing us to monitor the time at which the block appeared in the network and the time at which each connected node reported it to us.

[8] The observed variations in synchronization can also occur due to the rules deployed in the current Bitcoin RPC implementation shown in Figure 3. Since tailoring the default RPC implementation (*i.e.,* by not forwarding blocks to connected peers) can have ethical implications, therefore, we instead used the RPC implementation as provisioned in the protocol and reported results. Assuming that the RPC implementation results may vary from the actual network synchronization, that again highlights that measuring and mapping the network synchronization remains largely an open problem.

[9] Forks can be counted by (1) running a full node and executing *getchaintips* command, or (2) using the *ChainQuery*'s full node and executing *getchaintips* command [5].

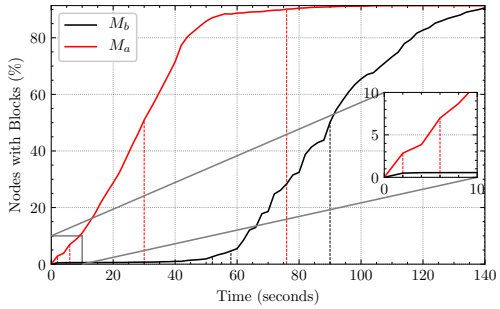


Figure 6: Block propagation pattern of two nodes (M_a and $M_b \in M$), sampled at one second interval. Note that M_a's block reaches $|M|/2$, $|M|$, $|\mathbb{N}|/2$, and $|\mathbb{N}|$ at 2, 6, 30, and 76 seconds, respectively. In contrast, M_b's block reaches the same set of nodes at 52, 58, 90, and 140 seconds respectively. M_a has a reachability advantage over M_b

.

In our second experiment, we observed that the mining nodes received blocks at different times, which clearly shows that the network is not *non-lock-step* synchronous in practice. Moreover, we also noted that the block propagation pattern for each mining node varied, demonstrating variations in their network reachability (*i.e.,* the number of connected peers). [10]

To further highlight the aforementioned observations, we present an example from our experiment in Figure 6, showing block propagation for two nodes (M_a and $M_b \in M$). Figure 6 shows that when M_a released the block, within 2 seconds, the block reached $|M|/2$, and within 6 seconds, the block reached $|M|$. Moreover, within 76 seconds, the block reached ≈90% of all the connected nodes $|\mathbb{N}|$. In contrast, when M_b released the block, the block took 52 seconds to reach $|M|/2$ and 58 seconds to reach $|M|$. The block took 140 seconds to reach ≈90% of all the connected nodes $|\mathbb{N}|$.

From the analysis above, we derived the following conclusions. (1) The current Bitcoin network is neither *lock-step* synchronous [13] nor *non-lock-step* synchronous [32, 40, 22]. (2) The block propagation pattern suggests that the mining nodes do not form a completely connected $M \times M$ topology.[11] The mining nodes topology 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 propagation suggest that the mining nodes have a varying network reachability. (4) Based on the observed network characteristics and the specifications provided in [34], the Bitcoin network can be considered asynchronous.

6 The *HashSplit* Attack

Nakamoto's consensus in a *non-lock-step* synchronous network increases the fork probability, wastes the effort of the honest miners, and lowers the cost for the majority attack [10, 32, 36]. Moreover, as indicated by Pass *et al.* [32], the problem becomes worse if the network is fully asynchronous, thus allowing an adversary to mount new attacks to violate the blockchain *safety* properties. Since our measurements indicate that the Bitcoin network is asynchronous, the next objective becomes formulating a new and feasible attack that violates the common prefix ($\mathcal{Q}_{cp}$) and the chain quality ($\mathcal{Q}_{cq}$) properties with high probability. Towards this objective, we present *HashSplit* which allows an adversary to exploit the asynchrony and orchestrate concurrent mining on multiple branches of a public chain to violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

HashSplit is a lower bound construction that shows: (1) an adversary with an arbitrary hashing power can violate $\mathcal{Q}_{cq}$, (2) an adversary with 26% hash rate can violate both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ with high probability, and (3) the requirement for a majority attack under any hash rate distribution can be amortized for all computationally admissible Bitcoin executions. From our measurements (§3–§5.1), we note that the asynchronous

[10] The observation regarding non-uniform block propagation patterns can also be verified on Bitnodes website, although without the knowledge of the mining nodes. The data on the website shows that some nodes report the block arrival in a few seconds. However, no two block propagation patterns are the same.

[11] Prior work [26] also reported that that mining nodes do not have a high network outdegree and they typically follow the standard node configurations. Our findings are consistent with the prior work.

Study	Requirement	Branches	Network Disruption
Decker *et al.* [10]	49% Hash Rate	Two or More	✗
Saad *et al.* [36]	30% Hash Rate	Two or More	✗
Natoli *et al.* [30]	5%	Two or More	✓
Apostolaki *et al.* [3]	–	One	✓
Heilman *et al.* [19]	–	One	✓
HashSplit	26% Hash Rate	Two or More	✗

Table 2: Comparing *HashSplit* with other attacks presented in prior works. Unlike [3, 30], the *HashSplit* adversary does not disrupt the network communication through a BGP attack. As such, with only 26% hash rate and concurrent mining on two or more blockchain branches, the adversary violates the blockchain *safety* and chain quality.

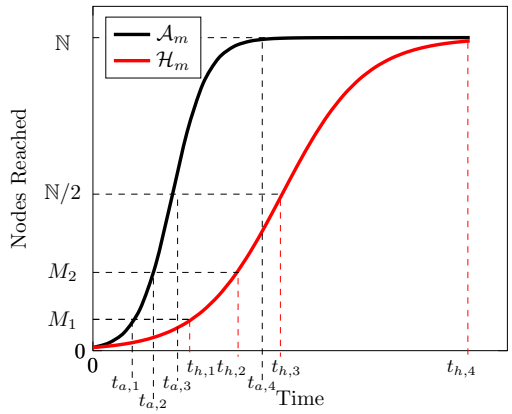


Figure 7: Generalized illustration of Figure 6. For simplicity of modelling, we assume a uniform hashing power distribution in M (*i.e.*, $M_1 \approx |M|/2 \approx 51\%$ hash rate and *i.e.*, $M_2 \approx |M|/2 \approx 49\%$ hash rate). $\mathcal{A}_m$ is well connected compared to $\mathcal{H}_m$. If $\mathcal{H}_m$ and $\mathcal{A}_m$ concurrently produce a block, $\mathcal{A}_m$ wins race due to $\mathcal{H}_m$'s propagation delay. Here $t_{a,1}, t_{a,2}, t_{a,3}$ and $t_{a,4}$ are times when $\mathcal{A}_m$'s block reaches 50% miners, 100% miners, 50% network, and 100% network. Accordingly, $t_{h,1}...t_{h,4}$ are the corresponding values for $\mathcal{H}_m$.

network creates a natural partitioning among the mining nodes which then expands the strategy space for an adversary to launch various attacks when combined with the Bitcoin mining policies [29, 32, 12, 10, 13].

It is worth noting that, unlike the balance attack [30] or the routing attacks [3, 36], *HashSplit* does not require the adversary to disrupt the network communication through BGP hijacking. Instead, the adversary simply exploits the existing propagation pattern 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. In Table 2, we compare *HashSplit* with similar attacks presented in the literature. For more details on *HashSplit* novelty, we refer the reader to §F.

6.1 Threat Model and Attack Objectives

For *HashSplit*, we assume an adversary $\mathcal{A}_m \in M$ with less than 51% hash rate. $\mathcal{A}_m$ follows the experiment methodology in §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 4), and obtain the block propagation pattern of each mining node (Figure 6). After identifying all $P_i \in M$, $\mathcal{A}_m$ maintains a direct connection with them to instantly send or receive blocks. Using the measurements, $\mathcal{A}_m$ calculates how a block generated by each $P_i \in M$ reaches all $P_i \in \mathbb{N}$. If $\mathcal{A}_m$ samples the block propagation pattern of each $P_i \in M$, Figure 6 can be expressed in terms of the general model in Figure 7. Note that Figure 7 is a generalized illustration that can be abstracted from any sequence of non-uniform block propagation pattern of two mining nodes, irrespective of the absolute delay.

Figure 7 shows that $\mathcal{A}_m$ has a strong network reachability like M_a in Figure 6, while $\mathcal{H}_m$ (an honest miner) has a weaker network reachability like M_b in Figure 6. Since Figure 6 is sampled at one second interval, $\mathcal{A}_m$ knows precisely 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, $\mathcal{A}_m$ can calculate the delay in the block reception for each $P_i \in \mathbb{N}$. For each $P_i \in M$, we define the *reachability time* $T_{i,j} = [t_{i,1}, t_{i,2}, t_{i,3}, t_{i,4}]$ as four time indexes at which the block is received by 50% miners, 100% miners, 50% network, 100% network.

We further assume that each $P_i \in M$, except $\mathcal{A}_m$, conforms to the ideal functionality such that when any $P_i \in M$ generates a block, it immediately releases the block to the network without withholding. Moreover, when a $P_i \in M$ receives two blocks with a hash pointer to the same parent block, $P_i \in M$ gives a 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 [29] and is currently deployed in all Bitcoin Core versions. Finally, we assume that (1) $\mathcal{A}_m$ cannot influence the communication model of other $P_i \in \mathbb{N}$ by launching routing attacks [3, 30], and (2) there is no other attack (*i.e.,* selfish mining) taking place concurrent with the *HashSplit* attack. 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 [3, 10, 36, 32].

Attack Objectives. Given that $\mathcal{A}_m$ is a miner with a view of the network's communication model, $\mathcal{A}_m$ can: (1) deviate from the ideal functionality and violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$, (2) waste the mining power of honest miners, and (3) prevent non-mining nodes from generating or receiving k-confirmed transactions [29]. In *HashSplit*, $\mathcal{A}_m$ achieves these goals by exploiting the block propagation pattern to split the public chain into two branches $\mathcal{C}_1$ and $\mathcal{C}_2$, and the mining nodes M into two groups M_1 and M_2. Here, M_1 is the group of miners mining on branch $\mathcal{C}_1$, while M_2 is the group of miners mining on branch $\mathcal{C}_2$. In a perfect split, $\mathcal{A}_m$ splits the network hash rate into $\mathcal{C}_1 \leftarrow \alpha = 0.51$ (mined by M_1), and $\mathcal{C}_2 \leftarrow \beta = 0.49$ ($\alpha + \beta = 1$) (mined by M_2, and mines on the branch with a higher hash rate. To violate $\mathcal{Q}_{cp}$ for any $P_i \in \mathbb{N}$, $\mathcal{A}_m$ ensures that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ for $k = 6$. To violate $\mathcal{Q}_{cq}$, $\mathcal{A}_m$ ensures that for any $P_i \in \mathbb{N}$, $\mu_i - \mu_i^{'} \neq \epsilon$ (the blockchain ledger has disproportionately high blocks mined by the adversary). In the following, we show that the *HashSplit* adversary meets these objectives with high probability.

6.2 Attack Procedure

6.2.1 Identifying Vulnerable Nodes

To split the blockchain, $\mathcal{A}_m$ first identifies the vulnerable mining nodes with a high reachability time by running algorithm 1. In algorithm 1, $T_{a,j}$ and $T_{i,j}$ are *reachability times* for $\mathcal{A}_m$ and other $P_i \in M$, respectively. $\mathcal{A}_m$ initializes four lists (aList...dList) and four variables (aMax...dMax). For each $P_i \in M$, $\mathcal{A}_m$ computes the time windows $\delta_1 ... \delta_4$ that represent the difference between the block propagation time of $\mathcal{A}_m$ and the target mining node. For intuition, we again refer to Figure 6, in which if assume M_a as $\mathcal{A}_m$ and M_b as $\mathcal{H}_m$, then algorithm 1 outputs $\delta_1 = 50$, $\delta_2 = 52$, $\delta_3 = 60$, and $\delta_4 = 64$ seconds, respectively. Therefore, algorithm 1 provides the difference in the *reachability time* of all $P_i \in M$ relative to $\mathcal{A}_m$'s *reachability time*. Additionally, algorithm 1 also determines the most vulnerable node with the maximum *reachability time* difference, which can be the easiest target to initiate the split.

6.2.2 Blockchain Splitting

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

Strategy 1. In this strategy $\mathcal{A}_m$ produces a block b_{r+1} before any $P_i \in M$, and withholds it. $\mathcal{A}_m$ waits for another $P_i \in M$ to produce a block $b_{r+1}^{'}$. With the apriori knowledge of $b_{r+1}^{'}$ propagation pattern in the network (algorithm 1), $\mathcal{A}_m$ releases b_{r+1} to M_1 while $b_{r+1}^{'}$ reaches M_2. As a result, when $b_{r+1}^{'}$ reaches M_1 after $t_{a,1}$, M_1 will not mine on it (time-based precedence [29]). However, by $t_{i,2}$, M_2 receive $b_{r+1}^{'}$ and start mining on it. Since the miners mine on the earliest received block (b_{r+1} for M_1 and $b_{r+1}^{'}$ for M_2), the blockchain forks into two branches $\mathcal{C}_1 \leftarrow \alpha$ and $\mathcal{C}_2 \leftarrow \beta$.

Strategy 2. In this strategy, an honest miner $P_i \in M$ produces the block $b_{r+1}^{'}$ before $\mathcal{A}_m$. Since $\mathcal{A}_m$ knows that $b_{r+1}^{'}$ will take $t_{i,1}$ time to reach M_1 (see Figure 7), $\mathcal{A}_m$ violates the ideal functionality and keeps mining for b_{r+1} until $t_{i,1}$. If $\mathcal{A}_m$ succeeds in mining b_{r+1} by $t_{i,1}$, $\mathcal{A}_m$ will release b_{r+1} to the other set of miners (M_2)

Algorithm 1: Identifying Vulnerable Mining Nodes

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

Algorithm 2: Attack Procedure (Split Ledger)

Input: M, $\mathcal{A}_m$
Case 1: $\mathcal{A}_m$ finds b_{r+1} before any $P_i \in M$
Strategy 1: $\mathcal{A}_m$ waits for another $P_i \in M$ to find b'_{r+1}. When $\mathcal{A}_m$ receives b'_{r+1} from $P_i \in M$, $\mathcal{A}_m$ releases b_{r+1} only to M_1 before M_1 receive b'_{r+1}. $\mathcal{A}_m$ does not release b_{r+1} to M_2, which invariably receive b'_{r+1} from the other miner at $t_{i,2}$ (Figure 7).
Case 2: Any $P_i \in M$ finds b'_{r+1} before $\mathcal{A}_m$
Strategy 2: $\mathcal{A}_m$ violates the ideal functionality (see **onStart** in Figure 1) and keeps mining b_{r+1}. By $t_{i,1}$, b'_{r+1} reaches M_1 miners. If $\mathcal{A}_m$ finds b_{r+1} before $t_{i,1}$, it releases b_{r+1} to M_2 before b'_{r+1} reaches them.
Result: In **Strategy 1**, M_1 receives b_{r+1} and M_2 receives b'_{r+1}. In **Strategy 2**, M_1 receives b'_{r+1} and M_2 receives b_{r+1}. In both cases, the chain $\mathcal{C}$ splits into two branches $\mathcal{C}_1$ and $\mathcal{C}_2$, and the network hash rate into α and β.

to which b'_{r+1} is yet to reach. As a result, and similar to **Strategy 1**, the blockchain splits into $\mathcal{C}_1 \leftarrow \alpha$ and $\mathcal{C}_2 \leftarrow \beta$. Therefore, algorithm 2 provides two strategies to split the chain into two branches.

Perfect Split. As described in §6.1, the perfect split leads to $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$. If $\mathcal{A}_m$, with a hash rate α_1, mines on $\mathcal{C}_1$, we define the combined hash rate of all miners in M_1 as $\alpha = \alpha_1 + \alpha_2$. $\mathcal{A}_m$ can achieve the perfect split since it knows the block propagation pattern and the hash rate distribution (Figure 7) of all the miners. $\mathcal{A}_m$ can time both strategies in algorithm 2 to achieve the perfect split such that $\alpha_1 + \alpha_2 = 0.51$.

Without losing generality, in the rest of the analysis we assume: (1) $\mathcal{A}_m$ achieves perfect split from algorithm 2, (2) there are four miners in the network ($\mathcal{A}_m$, h_1, h_2, and h_3), (3) $\mathcal{A}_m$ and h_1 mine on $\mathcal{C}_1$ with $\alpha_1 = 0.26$ and $\alpha_2 = 0.25$, (4) h_2 and h_3 mine on $\mathcal{C}_2$ with hash rates $\beta_1 = 0.25$ and $\beta_2 = 0.24$, respectively ($\beta = \beta_1 + \beta_2 = 0.49$), and (5) $\mathcal{A}_m$ has block propagation pattern similar to M_a in Figure 6 and all other miners have block propagation patterns of M_b in Figure 6. At $t_{a,1}$, $\mathcal{A}_m$'s block reaches h_1, and reaches both h_2 and h_3 at $t_{a,2}$. Similarly, for h_2, $t_{h,1}$ and $t_{h,2}$ are times at which $\mathcal{A}_m$ and both h_2 and h_3 receive a block. We can extend the same propagation sequence for h_2 and h_3.

We make these assumptions to simplify the analysis. The model can be easily generalized to more than four miners with varying hash rates and reachability times. The key idea is that as long as there is variation in block propagation patterns of the mining nodes (irrespective of the actual delay value), an adversary with better network reachability and faster block propagation can split the network and trigger concurrent mining on multiple branches of the public blockchain.

6.2.3 Block Race

Once the perfect split is achieved, the two chains, $\mathcal{C}_1$ and $\mathcal{C}_2$, enter in a block race. To formally analyze the race conditions, 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 $\mathsf{SHA256(SHA256((Header))} < d \in [0, 2^{256} - 1]$. On average, a miner computes $m = 2^{256}/d$ hashes to mine a block [17]. With the total network hash rate $\alpha + \beta$, $m = (\alpha + \beta) \times \tau$ is the total number of hashes required to mine a block at the specified block time τ [17]. When the hash rate is split into α and β (algorithm 2), the

time required to mine the next block on each branch becomes $t_o = m/\alpha$ and $t_o^{'} = m/\beta$. In other words, after executing algorithm 2, the next block from $\mathcal{C}_1$ is mined at t_o, and at $t_o^{'}$ for $\mathcal{C}_1$, respectively. Therefore, the probability that $\mathcal{C}_1$ succeeds in producing the block before $\mathcal{C}_2$ becomes $t_o/(t_o + t_o^{'}) = \alpha/(\alpha + \beta)$ [17, 18, 35]. Similarly, the probability that $\mathcal{A}_m$ mines the next block on $\mathcal{C}_1$ before h_1 is $\alpha_1/(\alpha_1 + \alpha_2)$, and the probability that h_1 mines the next block on $\mathcal{C}_1$ before $\mathcal{A}_m$ is $\alpha_2/(\alpha_1 + \alpha_2)$. This analysis can be easily extended to the miners h_1 and h_2 on the branch $\mathcal{C}_2$.

After executing algorithm 2, $\mathcal{A}_m$ needs to maintain the fork for k consecutive blocks to violate $\mathcal{Q}_{cp}$. 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 $\mathcal{A}_m$), $\mathcal{Q}_{cq}$ is violated. Note that since there are two public chains, if the fork gets resolved before k, and $\mathcal{C}_1$ is the winning chain, $\mathcal{Q}_{cq}$ is violated even when $\mathcal{Q}_{cp}$ is preserved. Considering these cases, in the following, we concretely specify the conditions under which the *HashSplit* attack succeeds or fails:

1. If the forks persist for more than k blocks, $\mathcal{Q}_{cp}$ is violated, and the attack succeeds partially.
2. If the forks get resolved before k blocks and $\mathcal{C}_1$ wins, $\mathcal{Q}_{cq}$ is violated, and the attack succeeds partially.
3. If the forks persist for k blocks and get resolved at $k+1$ block with $\mathcal{C}_1$ as the winning branch, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ 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 $\mathcal{A}_m$, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated. Moreover, in that case, the *HashSplit* attack becomes a majority attack since the adversary mines all blocks. In a synchronous network, the probability of this event is 0.08 with $\alpha_1 = 0.26$ [35].
5. If the forks get resolved before or after k blocks and $\mathcal{C}_2$ wins, $\mathcal{A}_m$ loses all blocks, and the attack fails.

Clearly, *HashSplit* relies on the block race outcomes in which the blockchain forks persist or get resolved. In Figure 8, we formally analyze all outcomes of a block race along with their probability distribution and $\mathcal{A}_m$'s strategies for the next round. We define a random variable X that specifies the probability distribution of the block race outcome in Figure 8. We further define $\mathbf{F}$ and $\mathbf{R}$ as the sum of events in which forks persist or get resolved. In (1) and (2), we show the probability $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$.

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

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

Plugging the hash rate of each miner from our threat model, $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$ become 0.6892 and 0.3108, respectively. From these values and Figure 8, we make the following conclusions.

1. With algorithm 2 as the starting point of a block race, there is higher probability that the given fork persists or new forks appear. This favors the violation of $\mathcal{Q}_{cp}$.
2. The probability that a fork is resolved by an honest miner on $\mathcal{C}_1$ is $\alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) = 0.1275$; significantly less than 0.6892 and favors $\mathcal{Q}_{cq}$'s violation.[12]
3. The probability that a fork is resolved by any honest miner on $\mathcal{C}_2$ is $\beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1) = 0.2401$. This is the failure probability for the attack, and it is considerably less than 0.6892 .

[12] If a fork is resolved by an honest miner, the adversary loses all blocks on the blockchain. Although, the probability of such an event is low (0.127).

Block Race

Fork Persists:

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

Fork Gets Resolved:

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

Figure 8: Block race after algorithm 2. For each event, we show the event probability and $\mathcal{A}_m$'s next strategy.

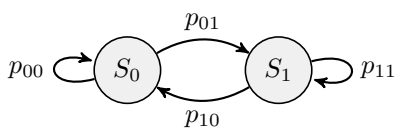


Figure 9: State machine representation of a block race. Transition probabilities are p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X=\mathbf{F}]$, $\mathbb{P}[X=\mathbf{R}]$, $\mathbb{P}[X=\mathbf{F}]$, and $\mathbb{P}[X=\mathbf{R}]$, respectively.

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 $\mathcal{Q}_{cp}$, and we show in Figure 8 how $\mathcal{A}_m$ can deal with more than two branches.

5. Block race can be modeled as a state machine in which the outcomes can be a fork with probability $\mathbb{P}[X_k = \mathbf{F}]$ or no fork with probability $\mathbb{P}[X_k = \mathbf{R}]$ [12, 24]. Figure 9 presents a state machine with S_0 and S_1 denoting states of forks and no forks, respectively. The transition probabilities p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X = \mathbf{F}]$, $\mathbb{P}[X = \mathbf{R}]$, $\mathbb{P}[X = \mathbf{F}]$, and $\mathbb{P}[X = \mathbf{R}]$, respectively.

6. Using Figure 9 and incorporating the propagation pattern, we can compute the long term probability of a forked blockchain that violates $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

Incorporating Propagation Advantage. Before computing the stationary distribution of Figure 9, it is important to incorporate $\mathcal{A}_m$'s mining advantage due to delay and block withholding. For instance, in $\mathbf{f}_1$, when $\mathcal{A}_m$ produces a block and withholds until h_2 or h_3 produce blocks, $\mathcal{A}_m$ can leverage the waiting time and the block propagation time to extend the newly mined block. The gap between $t_{a,1}$ and $t_{h,1}$ (or $t_{a,2}$ and $t_{h,2}$) provides additional time for $\mathcal{A}_m$ to mine the next block. To model this advantage, we first need to characterize the effect of delay on each miner's hash rate. Let $t_{a,0}$, $t_{h_{1,0}}$, $t_{h_{2,0}}$, $t_{h_{3,0}}$ be times at which $\mathcal{A}_m$, h_1, h_2, and h_3 mine blocks with hash rates α_1, α_2, β_1, and β_2, respectively. The relationship between propagation delay and the hash rate can be obtained as:

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

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

Considering $\alpha_1 = 0.26$, $\alpha_2 = 0.25$, $\beta_1 = 0.25$, $\beta_2 = 0.24$, and $\tau = 600$ seconds, from (3), $t_{a,0}$, $t_{h_1,0}$, $t_{h_2,0}$ become $\approx$ 2308, 2400, 2400, and 2500, respectively. Plugging these values in (4), the hash rate of each miner becomes $\alpha_1 = 0.259$, $\alpha_2 = 0.244$, $\beta_1 = 0.244$, and $\beta_2 = 0.235$. Next, to incorporate $\mathcal{A}_m$'s advantage in a block race, we convert δ_1 in algorithm 1 as the mining advantage that increases α_1. In our model, $\mathcal{A}_m$ gets ($\delta_1/\tau = 0.0833$ fraction of additional mining power. As a result, the *effective hash rate* of each miner becomes $\alpha_1 = 0.3423, \alpha_2 = 0.2163, \beta_1 = 0.2163$, and $\beta_2 = 0.2073$. Moreover, $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$ become 0.739 and 0.261, respectively.[13]

This advantage can be extended to miners when they resolve forks ($\mathbf{r}_1$ and $\mathbf{r}_2$ in Figure 8). 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 $\mathcal{A}_m$, and $t_{h,1}$ advantage over other miners. Empirically, $t_{a,1}$ accounts for $2/600 = 0.0033$, and $t_{h,1}$ accounts for $52/600 = 0.087$ fraction of the mining power. Therefore, if a fork resolve, the probability that it appears in the next round becomes $\mathbb{P}[X = \mathbf{F}] = 0.683$. Using these values, we can construct the transition probability matrix for Figure 9.

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

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

$$0.739\pi_1 + 0.261\pi_2 = \pi_1, \ 0.683\pi_1 + 0.317\pi_2 = \pi_2, \ \pi_1 + \pi_2 = 1 \tag{5}$$

From (5), $\pi_1 = 0.724$ and $\pi_2 = 0.276$, and the long term probability of a forked chain is significantly greater than of a single branch. Using the stationary distribution, we evaluate the impact of *HashSplit* on $\mathcal{Q}_{cp}$, $\mathcal{Q}_{cq}$, and the majority attack.

[13] Note that the delay provides a marginal incentive to the adversary in terms of the network hash rate. The attack would succeed even if the delay values are kept small (*i.e.,* 2–4 seconds, assuming a faster propagation). For instance, if we simply assume $\delta_1 = 2$ seconds, α_1, α_2, α_3, and α_4 become $\approx$ 0.263, 0.249, 0.249, and 0.239, respectively. Even in that case, $\mathbb{P}[X = \mathbf{F}]$ and and $\mathbb{P}[X = \mathbf{R}]$ remain reasonably high (0.69 and 0.31, respectively). Therefore, the key idea is as long as (1) mining nodes receive blocks at different times, and (2) the adversary (malicious mining node) has better network reachability than the honest mining nodes.

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

Chain Quality Property. Per (4), the block propagation affects the hash rate of each miner. As such, and even when not partitioning the blockchain, $\mathcal{A}_m$ can still mine more blocks that its hash rate allows. For instance, assuming an honest block race and $\delta_1 = 50$ seconds, $\mathcal{A}_m$ has $50/(3 \times 600)$ fraction of mining advantage over the other three miners ($\alpha_1 = 0.26, \alpha_2 = 0.223, \beta_1 = 0.223, \beta_2 = 0.213$). Moreover, if 100 blocks are mined, $\mathcal{A}_m$ will mine 28.29 blocks. From the ideal-world functionality view, $\mu - \mu^{'} = 2.29 \neq \epsilon$. $\mathcal{A}_m$ mines two blocks more than its hash rate, thus $\mathcal{Q}_{cq}$ is violated.

Common Prefix and Chain Quality. To violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$, a fork needs to persist or get resolved after k blocks, and $\mathcal{C}_1$ is the winning branch. Figure 8 shows that $\mathbf{r}_2$ is the only outcome where forks get resolved to $\mathcal{C}_2$ with probability 0.2401. We analyze that by branching S_1 in Figure 9 into two states and calculate the probability of $\mathcal{C}_2$ being the winning chain (computed as 0.167). Therefore, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated with a probability of $1 - 0.167 = 0.833$.

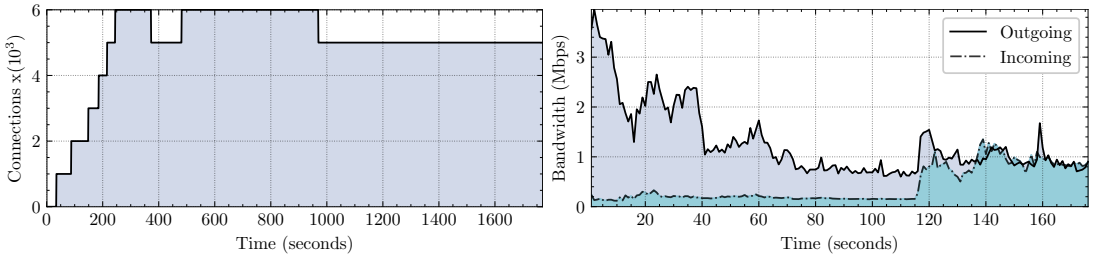
Majority Attack. From Figure 8, a majority attack happens if (1) $\mathcal{C}_1$ is the winning branch after k rounds, and (2) all blocks on $\mathcal{C}_1$ are mined by $\mathcal{A}_m$. This happens if for $k - 1$ rounds, one of the events $\mathbf{f}_i$, for $i = 1, 2, 3, 4, 7$ or 9 occurs, followed by event $\mathbf{f}_1$ on the k_{th} round. Similar to the analysis above, we can decompose this into a state machine where S_0 determines the probability of events $\mathbf{f}_i$, for $i = 1, 2, 3, 4, 7$ or 9, while S_1 determines the probability of $\mathbf{f}_i$ for $i = 5, 6$, or 8, and $\mathbf{r}_1$ or $\mathbf{r}_2$. From Figure 8, we compute p_{00}, p_{01}, p_{10}, and p_{11} as 0.663, 0.337, 0.576, and 0.424, respectively. For $k = 6$, the result is $(0.63 \times 0.342) = 0.2156$. Therefore, with a probability of 0.2156, *HashSplit* allows $\mathcal{A}_m$ 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 ≈ 0.08 [35].

In summary, *HashSplit* violates the blockchain *safety* properties with high probability and significantly lowers 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 weaker network synchronization compared to the mining nodes, making them more vulnerable to *HashSplit*. As noted in §1, splitting the mining power to lower the cost of the 51% attack is known in the literature [30, 15, 3]. 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 [27]). All these features make *HashSplit* more practical, stealthy, and feasible in the current Bitcoin network. We acknowledge that the asynchronous network can be exploited in several other ways to launch new attacks similar to *HashSplit* or further refine *HashSplit* by incorporating new strategies. However, covering all those attacks is beyond the scope of this paper. Moreover, since the Bitcoin network is permissionless and dynamic, the information propagation and blockchain synchronization can significantly vary with time (see [7]). However, irrespective of those changes, as long as the block propagation pattern of the mining nodes is different from each other (the network being asynchronous), the *HashSplit* attack can be launched by the adversary.

7 Attack Countermeasures

In this section, we discuss the attack countermeasures. Since *HashSplit* primarily exploits asynchronous network and block propagation pattern, if $\delta_1 \dots \delta_4$ in algorithm 1 are minimized, $\mathcal{A}_m$: (1) cannot split the mining nodes, and (2) cannot leverage a significant mining advantage. Additionally, if all $P_i \in M$ form $M \times M$ topology, Bitcoin will exhibit a *lock-step* or *non-lock-step* synchronous network which can be used to counter *HashSplit*.

In order to expedite block reception and form $M \times M$ network topology, we made a few refinements to Bitcoin Core in order to counter the attack [1]. We modified the source code to allow fast connectivity with Bitcoin nodes. From a mining node's perspective, the existing Bitcoin client may not offer a good network reachability to form $M \times M$ topology. For instance, it can take up to $\approx$120 days for all incoming connection slots to be full [38]. If those incoming connections include the mining nodes, it would rather be desirable to connect with them sooner in order to form the desired topology. For that purpose, we made refinements to



(a) Number of Bitcoin connections established in 1800 seconds

(b) Incoming and outgoing bandwidth consumption in Mbps.

Figure 10: Evaluation of our Bitcoin client deployed on a custom node. In less than 300 seconds, the node connected with over $6K$ reachable nodes while maintaining the bandwidth overhead under 6Mbps. The number of nodes drop around 900 seconds since Bitcoin Core automatically disconnects with peers that are *unresponsive* (*i.e.,* do not send *ping* or *pong* messages). Disconnecting such nodes saves connection slots and reduces the connectivity overhead [21].

Bitcoin Core by adding scripts that allow a faster connectivity [1].

For performance evaluation, we deployed our client on a local machine and evaluated the connectivity speedup and bandwidth consumption, with results reported in Figure 10. 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 ≈4Mbps. Our client is still in the testing phase and currently supports connections to IPv4 and IPv6 nodes.

From Figure 10, we note that a the node can connect to the mining nodes in less than 300 seconds. Through direct connectivity and better reachability, the node can instantly receive blocks from honest mining nodes, thereby minimizing $\mathcal{A}_m$'s advantage. However, we acknowledge that $M \times M$ topology does not fully counter the attack. Due to characteristics of the underlying Internet infrastructure (*i.e.,* low bandwidth), the network latency can be heterogeneous such that two peers connected to a same node can experience non-uniform propagation delay. Heterogeneous latency can be leveraged by $\mathcal{A}_m$ to launch the *HashSplit* attack even in $M \times M$ topology. Therefore, in addition to network layer remedies, we also require application layer defenses to counter the attack.

For application layer defenses, we equip our client with a *fork resolution* mechanism. We note from Figure 8, that the victim nodes have multiple branches of the same length in each round (*i.e.,* $\mathcal{C}_1$ and $\mathcal{C}_2$) during the attack. Particularly, miners on $\mathcal{C}_1$ will continuously receive blocks from $\mathcal{A}_m$, immediately followed by blocks from other honest miners. We leverage this sequence of block arrival to eliminate $\mathcal{A}_m$'s advantage and reduce the likelihood of a perfect split. In [1], 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 $\mathcal{A}_m$ will lose a direct connection to all mining nodes and will not be able to achieve a perfect split. $\mathcal{A}_m$ may deploy Sybil nodes in the network to connect to the victim. However, in that case $\mathcal{A}_m$ will lose δ_1 advantage over the victim since the block will be first relayed to the Sybil and then to victim node. Therefore, a combination of high network reachability and *fork resolution* mechanism can alleviate the risk of 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 functionality, identify the mining nodes, characterize the network synchronization, and present the effects of asynchrony on the Nakamoto consensus. We acknowledge that our measurement results can be improved and further enumerated to curate new attack strategies. However, the *HashSplit* attack is one demonstrative characterization of the execution model that is admissible in the Bitcoin computation model.

Limitations and Future Work. We also acknowledge some limitations in our work. First, due to limited resources, we could not conduct the measurements for a longer duration that could have provided

more interesting insights about the Bitcoin network. For instance, the per-second sampling of the network provided deeper insights into the synchronization pattern among the mining nodes. However, since it was highly storage-intensive, therefore, we only conducted it for a short duration to only validate asynchrony among the mining nodes, which is the fundamental idea behind *HashSplit*.

Conclusion. In this paper, we formulate the Bitcoin ideal functionality, identify the mining nodes, and show the network asynchrony in the real word. 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 and draws attention to the Bitcoin security properties. Moreover, our proposed countermeasures provide means to mitigate the attack by creating a *lock-step* synchronous network.

References

[1] Anonymous. Improved bitcoin core to counter hashsplit. https://anonymous.4open.science/r/56e77487-0470-4e10-b634-b13e939863c0/, 2020.

[2] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever. SABRE: protecting bitcoin against routing attacks. In *Network and Distributed System Security Symposium*. The Internet Society, 2019.

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

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

[5] ChainQuery. bitcoin-cli getchaintips – chainquery. https://chainquery.com/bitcoin-cli/getchaintips, 2020. (Accessed on 03/29/2021).

[6] B. Community. Six confirmation practice in bitcoin, 2019. https://en.bitcoin.it/wiki/Confirmation.

[7] B. Community. Bitnodes: Discovering all reachable nodes in bitcoin, 2020.

[8] M. Corallo. Bitcoin improvement proposal 152, 2018.

[9] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A. Sadeghi. Fastkitten: Practical smart contracts on bitcoin. In N. Heninger and P. Traynor, editors, *Security Symposium*, pages 801–818. USENIX, 2019.

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

[11] T. Duong, L. Fan, T. Veale, and H. Zhou. Securing bitcoin-like backbone protocols against a malicious majority of computing power. *IACR Cryptology ePrint Archive*, 2016:716, 2016.

[12] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[13] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology*, pages 291–323. Springer, 2017.

[14] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer. Decentralization in bitcoin and ethereum networks. *CoRR*, abs/1801.03998, 2018.

[15] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[16] S. Goldberg and E. Heilman. Technical perspective: The rewards of selfish mining. *Commun. ACM*, 61(7):94, 2018.

[17] C. Grunspan and R. Pérez-Marco. Double spend races. *CoRR*, abs/1702.02867, 2017.

[18] C. Grunspan and R. Pérez-Marco. On profitability of selfish mining. *CoRR*, abs/1805.08281, 2018.

[19] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.

[20] J. Jang and H. Lee. Profitable double-spending attacks. *CoRR*, abs/1903.01711, 2019.

[21] JBaczuk. bitcoind - specific explanation of "timeout" configuration option in bitcoin - bitcoin stack exchange. https://bit.ly/3tWbJCg, 2021. (Accessed on 03/25/2021).

[22] L. Kiffer, R. Rajaraman, and A. Shelat. A better method to analyze blockchain consistency. In *Conference on Computer and Communications Security*, pages 729–744, 2018.

[23] Y. Kwon, D. Kim, Y. Son, E. Y. Vasserman, and Y. Kim. Be selfish and avoid dilemmas: Fork after withholding (FAW) attacks on bitcoin. In *Conference on Computer and Communications Security*, pages 195–209. ACM, 2017.

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

[25] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun. BITE: bitcoin lightweight client privacy using trusted execution. In *Security Symposium,*, pages 783–800. USENIX, 2019.

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

[27] A. Mining. Antpoolhashrate, 2020.

[28] R. Nagayama, R. Banno, and K. Shudo. Identifying impacts of protocol and internet development on the bitcoin network. In *Symposium on Computers and Communications*, pages 1–6. IEEE, 2020.

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

[30] C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *International Conference on Dependable Systems and Networks*, pages 579–590. IEEE, 2017.

[31] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh. Erlay: Efficient transaction relay for bitcoin. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *Conference on Computer and Communications Security*, pages 817–831. ACM, 2019.

[32] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.

[33] Poolin. Pool stats bitcoin mining pools. https://btc.com/stats/pool, 2021. (Accessed on 03/29/2021).

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

[35] M. Rosenfeld. Analysis of hashrate-based double spending. *CoRR*, abs/1402.2009, 2014.

[36] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen. Partitioning attacks on bitcoin: Colliding space, time, and logic. In *International Conference on Distributed Computing Systems*, pages 1175–1187. IEEE, 2019.

[37] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.

[38] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang. A stealthier partitioning attack against bitcoin peer-to-peer network. In *Symposium on Security and Privacy*, pages 894–909. IEEE, 2020.

[39] C. Wang, X. Chu, and Q. Yang. Measurement and analysis of the bitcoin networks: A view from mining pools. *CoRR*, abs/1902.07549, 2019.

[40] J. Zhao, J. Tang, Z. Li, H. Wang, K. Lam, and K. Xue. An analysis of blockchain consistency in asynchronous networks: Deriving a neat bound. In *International Conference on Distributed Computing Systems*, pages 179–189, 2020.

A Ideal World Functionality Proof

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

Theorem 1 (Bitcoin Ideal World Functionality)**.** *If the protocol is run for l= 6 consecutive rounds, in which k = 6 blocks are produced, then with 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 [6]. Moreover, Theorem 1 assumes that in each round, only one block is produced, and therefore, for l consecutive rounds, a total of $k = l$ blocks are produced.

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

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

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

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

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

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

B 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.* [13] 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.* [32] that analyzes the performance of Blockchain protocols in asynchronous networks.[14]

The Bitcoin Backbone Protocol [13]. 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* [34]. Moreover, the model assumes $M = \mathbb{N}$, where the hash rate is uniformly distributed among all miners. Finally, the adversary does not control more than $|M|/2 = |\mathbb{N}|/2$ miners. In other words, the miner is bounded by 50% hash rate. Using these assumptions, the [13] 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 defined as the security parameter.*

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

Although the Bitcoin backbone protocol in [13] formally specifies the properties of the Bitcoin system, however, it makes some assumptions that deviate from the real world implementation. In the following, we briefly discuss them.

(1) Firstly, the model assumes $M = \mathbb{N}$ and the mining power to be uniformly distributed. However, as shown in §4, $M << \mathbb{N}$ and the mining power is not uniformly distributed. (2) Secondly, the synchronous execution assumes that in each round, the block experiences no propagation delay. As a result, the adversary gains no advantage from the propagation delay. In our measurements, we have observed that the Bitcoin network is not a fully 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 [13], this means that if the adversary controls $\approx |M|/4$ or $|\mathbb{N}|/4$ miners, the common prefix property and the chain quality property will not hold.

Blockchains in *non-lock-step* Synchronous Networks [32]. Improving the model of Garay *et al.* [13], Pass *et al.* [32] proposed an *non-lock-step* synchronous model to evaluate Bitcoin. Their proposition introduces a network delay parameter Δ that an adversary can add during block propagation. By the time the block reaches other miners, the adversary leverages Δ to gain a head start mining advantage towards computing the next block. In the following, we analyze the model proposed in [32].

(1) The primary assumption of their model is that an adversary is able to compute a block, delay its transmission by an upper bound Δ, and broadcast it to the network. After the broadcast, all participants receive the block and the *non-lock-step* synchronous network starts to *emulate* the *lock-step* synchronous network [13]. Therefore the key difference in the *non-lock-step* synchronous model [32] and the *lock-step* synchronous model [13] is Δ, after which both models *emulate* the same behavior. (2) Δ gives an advantage to the adversary over all the other participants. Roughly speaking, in the Δ time window, the adversary is able to mine on top of its previous block. Moreover, [32] assumes that during Δ, all the other participants remain idle. More formally, the model specifies $\alpha \approx p(1-\rho)n$ to be the probability that an honest player computes the block. Here, p is the mining hardness and ρ is the fraction of nodes in n that the adversary controls. Moreover, $\beta = \rho np$ is the expected number of blocks that an adversary can mine in a round. (3) Once the bounded delay Δ is plugged into the system, the model in [32] assumes a parameter $\delta > 0$ such that to meet consistency property, the model has to satisfy $\alpha(1-(2\Delta+2)\alpha) \geq (1+\delta)\beta$. As long as $\rho < 0.5$ and $p < \frac{1}{pn\Delta}$, consistency is satisfied. (3) The bounded delay Δ also "*discounts*" the capability α of honest nodes to produce blocks in a round. To formally capture that, the model in [32] introduces $\gamma = \frac{\alpha}{1+\Delta\alpha}$ which is the discounted version of α or the effective mining power gained by delaying the block propagation by Δ.

[14] Although Pass *et al.* call their model asynchronous, however, Ren *et al.* [34] show that their model is actually *non-lock-step* synchronous. For simplicity, in this section, we still refer to [32]'s model as asynchronous.

Using the aforementioned assumptions and a security parameter κ, [32] proposes two theorems to characterize Nakamoto consensus specific to the Bitcoin operations.

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

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

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

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

Then, except with an exponentially small probability (in T), Nakamoto consensus satisfies T-consistency under the assumption that the network latency is bounded by Δ.

The consistency property in (9) can also be interpreted as the common prefix property in [13]. The T-consistency specifies that the two ledgers must share a common prefix after pruning the last T blocks from their chains ($\mathcal{C}_1^{\lceil T} \preceq \mathcal{C}_2$). Moreover, in [32] (Section 3.5), the authors mention "for instance, in the Bitcoin application, we are interested in achieving T-consistency for $T = 6$." This is similar to our formulation of the ideal world functionality and its proof, where we prove that ($\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, for $k = 6$). However, in the *HashSplit* attack, we show that the adversary violates the consistency property by deviating from the ideal world functionality. Additionally, we provide Theorem 6 to improve the characterization of [32], where the *non-lock-step* synchronous model starts to emulate the properties of the *lock-step* synchronous model.

Experimental Interpretation. The experimental setup in [32] assumes a network where each node can mine. Moreover, it also assumes $n = 10^5$ and $\Delta = 10s$. The results show that Nakamoto consensus tolerates a 49.75% attack, under $\Delta = 10s$ bound.

Non-lock-step Synchronous Model in Real World

Although the *non-lock-step* synchronous model in [32] 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, [32] assumes that during Δ, other mining nodes remain idle, and after Δ, the system abruptly starts emulating *lock-step* synchronous behavior. This would imply that when an adversary delays a block by Δ, no other miner receives the block in the meantime, and after Δ, all miners receive the block instantly and start mining on top of it. This generalization does not capture the real world Bitcoin operations. For instance, assume there are $w = |M|$ mining nodes in the system. Each node is $1, ..., w$ hops away from a typical mining node (adversary in this case). Further, assume that the adversary releases the block and the block incurs a delay at each hop. Now, $P_1 \in M$ receives the block after Δ_1, $P_2 \in M$ receives the block after Δ_2, and $P_w \in M$ receives the block after Δ_w. Naturally, $\Delta_1 < \Delta_2 < \Delta_w$. In PoW, each miner is motivated by its interests. If the miner receives a new block, for which it has been unsuccessfully mining, it immediately drops its computation and starts mining on top of the newly received block. Therefore, the network does not exhibit synchronous behavior until all $|M|$ miners receive the block. Moreover, if by the time $P_w \in M$ receives the block, and another miner $P_1 ... P_{w-1} \in M$ produces the next block, then the system may never exhibit the synchronous model. For a synchronous execution, all miners in M should be mining for the same block. Acknowledging this requirement for a synchronous execution, in the following, 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. □

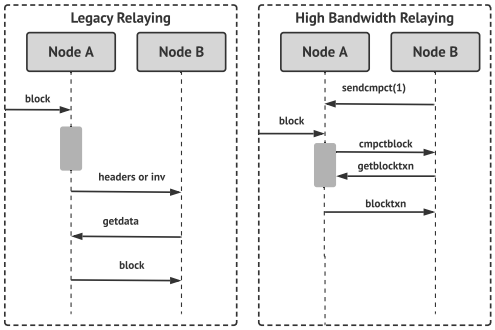


Figure 11: Differences between INV-based block relaying and CMPCTBLOCK-based block relaying. In INV relaying, when Node A receives a block, first it verifies the contents of the block and then issues INV message to which Node B responds with a GETDATA message. In CMPCTBLOCK relaying, first Node B sends a SENDCMPCT message to Node A to signal that it supports CMPCTBLOCK relaying. As a result, when Node A receives a block, it immediately forwards it to Node B without validating headers. Node B then reconstructs the block from its mempool.

Note 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 [32]. They will be mining on top of b_r. Therefore values of α, γ, and δ in Theorem 4, Theorem 5 will change. In the case that we have outlined above, α will become $\alpha_1, ..., \alpha_w$, and similarly δ and γ would change to $\delta_1, ..., \delta_w$ and $\gamma_1, ..., \gamma_w$, respectively. The value of β will however remain unchanged. As a result, in the real world settings, the overall advantage of the adversary, due to Δ will decrease. Plugging this into [32], 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.* [10]. 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.

C Data Collection Details

At each crawler, we used high-speed fiber-optic Internet with a 1Gbps connection. After five weeks, we discontinued the experiment since we we had sufficient data (*i.e.,* information about the mining nodes and variations in block propagation) to motivate for the *HashSplit* attack. We also 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 the INV message).

D Revisiting Partitioning Attacks

In this section, we revisit the two notable partitioning attacks proposed in [36, 3] to understand how the new insight we uncovered about the mining nodes (§4) refines those attacks.

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

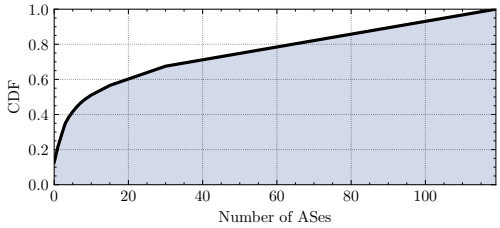


Figure 12: Distribution of mining nodes across ASes. Compared to the general distribution of full nodes [36, 3], mining nodes are comparatively more decentralized, and therefore, less vulnerable to routing attacks. 90 (31.69%) mining nodes are uniquely hosted across 90 ASes (at the time of this study), showing a higher distribution in the hosting patterns.

Routing Attacks [3]. 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 [30]. 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 12, 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 [2, 36]. 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.

E Network Synchronization

In the following, we contrast our findings against prior works to highlight the aspects of weak network synchronization.

In 2012, Decker *et al.* [10] connected to ≈3.5K *reachable* nodes and observed that 90% of them received a new block in 12.6 seconds. They also observed a 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 2019, Saad *et al.* [36] observed a weaker network synchronization than [10] using Bitnodes dataset. They attributed the change in synchronization to the increasing network size.

Our measurements also show weak network synchronization with ≈39% nodes having an up-to-date blockchain. After observing a surprisingly weak synchronization, we followed up our analysis by observing the Bitcoin network synchronization through Bitnodes [7]. Bitnodes provides APIs to (1) observe the latest Bitcoin block, and (2) latest block on the tip of all Bitnodes' connected nodes. The gap between the latest block on the Bitnodes blockchain and the latest block on the blockchain tip of Bitnodes' connected nodes can be used to estimate the network synchronization.

We conducted our experiment from October 2020 to December 2020 and found that at any time, only 52.2% nodes had an up-to-date blockchain, with the maximum and minimum values of 86% and 15% respectively.[15]

Variations in network synchronization can also occur due to the current Bitcoin RPC protocol implementation shown in Figure 3. Since tailoring the default RPC implementation (*i.e.,* by not forwarding blocks to the connected peers) can have ethical implications, therefore, we instead used the RPC implementation as provisioned in the protocol and reported the results in Figure 5. Assuming that the RPC implementation

[15]Variations between our results and Bitnodes (Figure 5) could be due to the difference the time of conducting the experiments and the methodology of network sampling. In our experiments, we measured the Bitcoin network synchronization immediately after receiving a new block. (1) The synchronization pattern observed is significantly below the expected value specified in the ideal functionality Figure 1. (2) Variation in the synchronization pattern (see Figure 13) is itself indicative of an asynchronous network which is the foundation of the *HashSplit* attack. (3) Since Bitnodes data is publicly available, we encourage future research towards analyzing the gaps in network synchronization and improving them.

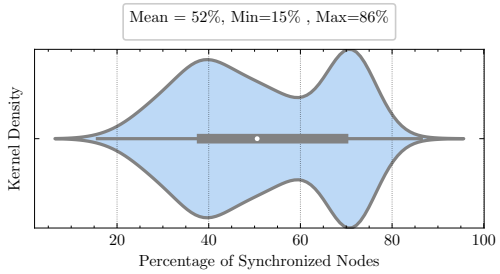


Figure 13: Network synchronization observed in the Bitnodes dataset. The results show that on average, only 52% nodes have an up-to-date blockchain at any time, with the maximum and minimum values of 86% and 15% ,respectively. The data distribution shape (Kernel Density) is itself indicative of an asynchronous network since there are variations in block propagation and network synchronization.
results may vary (*i.e.,* due to protocol implementation) from the actual network synchronization, that again highlights that measuring and mapping the network synchronization remains largely an open problem.

F 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 [12, 11]. The 51% attack primarily relies on the ability to generate the "longest chain" in the long run [4]. To understand how the majority attack works, assume an attacker with the hash rate h_a, participating in the block race. The attacker is s blocks behind the rest of the network and aims to catch up with a private chain that is longer than the public chain. If the rest of the network with hash rate $H - h_a$ finds the next block then the attacker will be $s+1$ blocks behind, with his success probability as a_{s+1}. Conversely, if the attacker finds the next block with probability, the gap will reduce by $s-1$, with his success probability as a_{s-1}. Given this information, a_s must satisfy the following recurrence relationship.

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

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

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

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

Selfish Mining. Selfish mining is a form of attack, in which the adversary computes a block and does not publish it [35]. Instead, it keeps on extending its private chain in hopes that the honest miners on the public chain invariably switch to the adversary's private chain. The chain switching by honest miners invalidates their effort of the honest miners. The selfish mining strategy proposed by [12] undermines the incentive compatibility of Bitcoin and can be launched with less than the majority hash rate.

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,

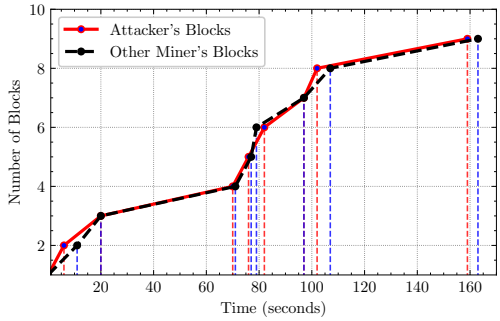


Figure 14: Simulations of the *HashSplit* attack. In each round (except 5th), the adversary with 26% hash rate is the first to produce a block and follows algorithm 2. In the 5th round, the adversary manages to produce the block before $t_{h,2}$. Adversary releases the chain after 8th block

and extends that chain. If the private chain becomes longer than the public chain (with probability 1 if the attacker has 51% hash rate), then the recipient's transaction will be invalidated.

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

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

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

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

G Simulations

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 some delay. This strategy allowed us to construct the network topology without incurring the overhead of a client-server socket implementation.

Figure 14 shows that except for the 5th block, the adversary is able to find a block before any other miner. After computing the block, the adversary waited for any other miner from the competing chain to release the block while extending its chain atop its previously mined block. In our results, we observed that at the 5th block, a miner on the second public chain produced a 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, 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.