Revisiting Nakamoto Consensus in Asynchronous Networks: A Comprehensive Analysis of Bitcoin Safety and Chain Quality

Muhammad Saad, Afsah Anwar, Srivatsan Ravi, *Member, IEEE*, and David Mohaisen, *Senior Member, IEEE*

***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 feasibly 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 103K IP addresses of the Bitcoin nodes and identify 871 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 Bitcoin and similar blockchain systems.**

***Index Terms*— Nakamoto consensus, Bitcoin partitioning.**

I. Introduction and Related Work

BITCOIN is a dynamically evolving distributed system that has significantly scaled up in recent years [2]. As Bitcoin continues to grow and inspire other decentralized applications, its security features are continuously investigated using theoretical analysis and measurement techniques [3], [4]. 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.

A. Theoretical Models' Shortcomings

The existing theoretical models [5], [6], [7], [8] that formally analyze the Nakamoto consensus (1) sidestep 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 [9], [10].[1] To incorporate the mining centrality in a theoretical model, we construct the Bitcoin ideal functionality (§II), which acknowledges the distinction between the mining and non-mining nodes and presents an execution model that preserves the blockchain *safety* properties.

Another limitation of the existing theoretical models is that they assume uniform block propagation delay patterns. The Bitcoin backbone protocol, proposed by Garay et al. [6], assumes a *lock-step* synchronous network in which the adversary does not benefit from the block propagation delay. This assumption is impractical for a large-scale distributed system such as Bitcoin, where block propagation incurs non-zero delay [11]. To address this limitation, Pass et al. [7] extended the work in [6] and analyzed Bitcoin in the *non-lock-step* synchronous settings [8]. The *non-lock-step* synchronous model assumes a network in which all miners experience the same block propagation delay, giving a uniform advantage to the adversary over all other miners. Our measurements provide a different view of the network as we observe that miners receive blocks at different times thereby indicating that the network is neither *lock-step* nor *non-lock-step* synchronous. In fact, the network exhibits a notion of asynchrony whereby block reception at different times can influence the mining strategy of the nodes. Furthermore, by exploiting the potential change in

Manuscript received 20 August 2022; revised 24 January 2023; accepted 9 July 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor N. Zhang. Date of publication 21 August 2023; date of current version 16 February 2024. An earlier version of this work has appeared in CCS 2021 [DOI: 10.1145/3460120.3484561]. The new manuscript advances upon the previous version with new datasets and more accurate evaluation. This work was supported in part by NRF under Grant 2016K1A1A2912757. *(Corresponding author: Muhammad Saad.)*

Muhammad Saad, Afsah Anwar, and David Mohaisen are with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: saad.ucf@knights.ucf.edu).

Srivatsan Ravi is with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089 USA.

Digital Object Identifier 10.1109/TNET.2023.3302955

[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 [9] and [10], 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 nodes.

mining strategies, an adversary can frequently fork the chain. In §V, 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 information propagation which is pertinent to ensure the two blockchain *safety* properties, namely the common prefix and the chain quality [6], [12]. In §VI, we show that the asynchronous network relaxes the requirement to violate these two properties.

B. 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) [9], [13], [14], (2) discovering influential nodes controlled by the mining pools [9], [10], and (3) measuring the information propagation [11]. The security evaluations of these studies proposed (1) partitioning attacks through BGP prefix hijacking of high profile ASes [9], (2) majority attacks with less than 51% hash rate (≈49% in [11]), and (3) a combination of the two attacks (*i.e.,* the balance attack in [15]).

In 2012, Decker and Wattenhofer [11] conducted the first measurement study to analyze the block propagation pattern in the Bitcoin Peer-to-Peer (P2P) network. They reported a block propagation delay of ≈11 seconds after which 90% of all the *reachable* nodes received a newly mined block. Using the *non-lock-step* synchronous model, they derived a relationship between the block propagation delay and the *effective* network hash rate, concluding that the block propagation delay of 11 seconds reduces the majority attack requirement to 49% hash rate. Our measurements validate the initial premise of block propagation delay in [11], and further demonstrate that the mining nodes receive blocks at different times. As a result, we conclude that the Bitcoin P2P network cannot be characterized as a *non-lock-step* synchronous network with uniform delay experienced by the mining nodes.

C. Limited Attack Strategies

The attacks proposed in prior studies have not been frequently observed in the wild due to strong adversarial requirements and limited attack strategies. First, their threat models directly inherit the assumptions of theoretical frameworks in [6] and [7] and ignore the critical distinction between the mining and non-mining nodes (*i.e.,* in [11], [13], and [15]). 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 [7]. The key challenge lies in getting visibility into the network intricacies to (1) identify the mining nodes, (2) study the block propagation pattern, 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.

D. 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 [9], the authors show that BGP attacks can reduce the mining power of the Bitcoin network. In [15], Natoli and Gramoli 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. [16]). Similarly, the Eclipse attack [17] 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 [18]. Therefore, they implicitly assume a form of route manipulation (*i.e.,* BGP hijacking or occupying incoming and outgoing connections [15], [17]) 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.[2]

E. 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. We model the attack on an adversary with 26% mining power and demonstrate that with less than the majority hash rate (51%), the adversary can still violate the blockchain *safety* properties by exploiting the non-uniform blockchain propagation pattern. 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 which 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 [6] (§II). The ideal world functionality faithfully models the expected functionality of a correct Bitcoin implementation across prevalent deployments in real world Bitcoin network.
2) We deploy crawlers in the Bitcoin network and connect with over 103K IP addresses of Bitcoin nodes in 47 days (§III). Using heuristics, we identify 871 IP addresses of the mining nodes. (§IV).

[2]Note that *HashSplit* does not rely on the absolute delay in the block propagation as used previously in [11]. Instead, we show that as long as mining nodes receive blocks at different times, irrespective of the absolute delay, the adversary can exploit the protocol specifications in order to partition them and violate the blockchain *safety* and chain quality with a high probability.

3) We measure the block propagation patterns in the Bitcoin network, and through a fine-grained analysis of the block propagation patterns, we show that execution of the Nakamoto consensus in Bitcoin is asynchronous §V.
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 [19].

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 keeping with the flow, this paper first introduces the ideal world functionality followed by the measurements, attack, and conclusion.

II. 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 decentralized network where any node in the network could compete to mine a block (1 CPU=1 Vote) [5]. However, over the years, Bitcoin mining became more competitive, with large mining pools becoming the key players in the network that mined most of the blocks [20]. While mining centralization has become a reality in the Bitcoin network, we note that the existing theoretical models do not incorporate this change in their analysis of the Bitcoin network.

In our ideal world functionality, we embrace the practical aspects of the Bitcoin network to distinctly characterize the role of mining and non-mining nodes. We define mining nodes as nodes that are owned by the mining pools and are responsible for mining blocks. On the other hand, the non-mining nodes use those blocks for transaction validation. Note that the formulation of our ideal functionality is inspired by theoretical models proposed in [6] and [7], 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 [6].

First, we define $\mathbb{N}$ as the set of all reachable IP addresses of Bitcoin nodes. We define $\text{VIEW}_{\mathcal{C}}^{P_i}$ as the blockchain view of a single node $P_i \in \mathbb{N}$, where $\mathcal{C}$ is the blockchain ledger. The Bitcoin backbone protocol [6] states that the inter-arrival time between two blocks must be sufficiently large that each $P_i \in \mathbb{N}$ has $\text{VIEW}_{\mathcal{C}}^{P_i}$ (*i.e.,* in ≈10 minutes, all $P_i \in \mathbb{N}$ have the up-to-date blockchain). Next, we define $\{M \subset \mathbb{N}\}$ as a set IP addresses of the mining nodes.[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.

[3] $M = \mathbb{N}$, implies all nodes are the mining nodes. However, in §IV, we show that due to mining centralization, there are only 871 mining nodes among 103K IP addresses ($|M|$=871 and $|\mathbb{N}|$=103K).

TABLE I
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

Model	Network	Topology	Mining Nodes
Garay *et al.* [6]	*lock-step* synchronous	Strongly Connected	✗
Pass *et al.* [7]	*non-lock-step* synchronous	Strongly Connected	✗
This Work	*lock-step* synchronous	Strongly Connected	✓

A. 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 $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$. $\mathcal{C}_1^{\lceil k}$ 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 $k = 6$) [21].

B. 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.[4] $\mathcal{Q}_{cq}$ ensures that for a sufficiently large value of l, the contribution of P_i in $\mathcal{C}$ is proportional to its hash rate h_i. Moreover, $\mathcal{Q}_{cq}$ assumes that no $P_i \in M$ acquires more than 50% hash rate [22], [23], [24], [25], [26].

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 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 [8].

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. The mining nodes participate in the block race, while the non-mining nodes simply maintain the blockchain ledger. Moreover, when a mining node receives two blocks in a round, referencing the same parent block, it gives time-based precedence to the block received earlier and starts extending the chain on that block. Note that if the network does not form a fully connected topology as specified in the ideal functionality, the network will deviate from the expected behavior of a *lock-step* synchronous network. As a result, nodes may receive a new block at different times. If two blocks are mined in a round, the mining nodes will mine on the block that they receive first, leading to concurrent mining on two

[4] Honest blocks refer to blocks mined by the honest nodes that faithfully follow the Bitcoin protocol specifications.

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 mines on b_{r+1} and treats $b_{r+1}^{'}$ 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.

Fig. 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.

branches of the public chain. We will later show in §V, how the real world network deviates from the ideal functionality, thereby creating opportunities to launch the *HashSplit* attack. In Table I, we contrast our ideal world functionality against the prior theoretical models and in the following, we provide the ideal world functionality proof.

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

Theorem 1 (Bitcoin Ideal World Functionality): If the protocol is run for $l = 6$ consecutive rounds, in which $k = 6$ blocks are produced, then with 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 [21]. 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:

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

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 (1)) becomes:

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

With $\lambda = 1/600$, $k = 6$, and $x = 2$, $P(X)$ is 0.00001. In other words, the ideal world functionality guarantees the common prefix for $k = 6$ with an 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 (μ_i'. Therefore, $\mu_i - \mu_i'$ is 0.001. This is negligible (ϵ) as defined in the ideal world functionality Figure 1. ■

In the following, we elaborate on prior theoretical models in [6] and [7] by analyzing their key model primitives in the context of our ideal functionality.

The first model is the foundational work proposed by Garay et al. [6] which provides the formal definitions of the "Common Prefix Property" and the "Chain Quality Property." The second model is another prominent work proposed by Pass et al. [7] which analyzes the performance of Blockchain protocols in asynchronous networks.[5]

C. The Bitcoin Backbone Protocol [6]

Garay et al. proposed a synchronous network in which when a miner releases a block, the block is received by all other miners without delay. Therefore, the protocol is run in a *lock-step* synchronous network [8]. 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 parameters, [6] proposes two theorems for common prefix and chain quality properties.

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 [6] 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) First, the model assumes $M = \mathbb{N}$ and the mining power to be uniformly distributed. However, as shown in §IV, $M \ll \mathbb{N}$ and the mining power is not uniformly distributed. (2) Second, the synchronous network assumes that in each round, the block experiences no propagation delay. As a result, the adversary gains no advantage from the propagation delay. Our measurements showcase a different network state where mining power is centralized and block propagation incurs delay. Moreover, block propagation can vary in each execution round, which can be exploited by an adversary to create forks.

[5]Although Pass et al. call their model asynchronous, however, Ren [8] show that their model is actually *non-lock-step* synchronous.

D. Blockchains in Non-Lock-Step Synchronous Networks [7]

Improving the model of Garay et al. [6], Pass et al. [7] 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 [7].

(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 [6]. Therefore the key difference in the *non-lock-step* synchronous model [7] and the *lock-step* synchronous model [6] 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, [7] 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 [7] 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 [7] introduces $\gamma = \frac{\alpha}{1+\Delta\alpha}$ which is the discounted version of α or the effective mining power gained by delaying the block propagation by Δ.

Using the aforementioned assumptions and a security parameter κ, [7] proposes two theorems to characterize Nakamoto consensus specific to the Bitcoin operations.

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

$$\mu_{\delta}^{p}(\kappa, n, \rho, \Delta) = 1 - (1+\delta)\frac{\beta}{\gamma} \tag{3}$$

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

$$\alpha(1-(2\Delta+2)\alpha) \geq (1+\delta)\beta \tag{4}$$

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 (4) can also be interpreted as the common prefix property in [6]. 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 [7] (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.

E. Key Takeaways

Theoretical models proposed by Garay et al. [6] and Pass et al. [7] have generated an ecosystem impact by significantly enhancing our understanding of the Bitcoin network. Although our ideal world functionality is inspired from their models, we make an effort to embrace practical aspects of Bitcoin implementation. For that purpose, our ideal world functionality incorporates mining and non-mining nodes as well as their actions when presented with varying protocol conditions (*i.e.,* select the earliest received block if more than one block is mined in a round).

In the rest of the paper, we perform a data-driven study to investigate the size $|M|$ of the mining nodes and the block propagation patterns in the network. Analyzing these two aspects is critical for the following reasons. Discovering the size of mining nodes $|M|$ will justify the need to create a distinction between mining and non-mining nodes in our ideal world functionality. The difference between $|\mathbb{N}|$ and $|M|$ will empirically show the degree of mining centrality in the current network. Moreover, variation in the block propagation pattern of the mining nodes M is necessary to highlight network asynchrony.

III. 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.

A. 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 [10]) only establish outgoing connections. For simplicity, we can characterize the Bitcoin network between the *reachable* space and the *unreachable* space, as shown in Figure2.

It is argued that mining pools prefer to host their mining nodes in the *unreachable* space due to security concerns. 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 §VI. 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 [11]. 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.

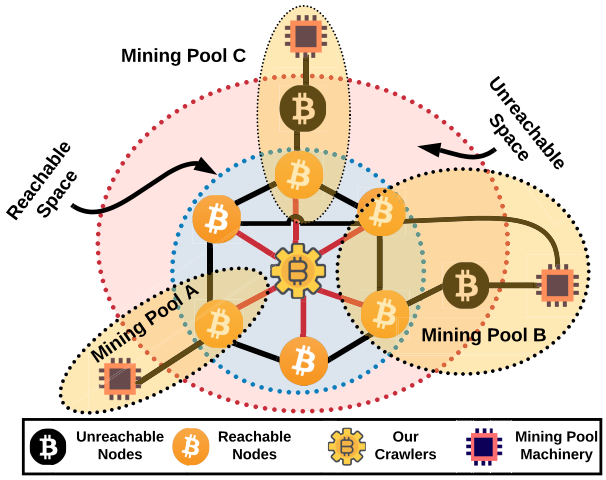


Fig. 2. An illustration of our data collection system contextualized in the Bitcoin network. Mining pools can have *reachable* (Mining Pool A), *unreachable* (Mining Pool C), or both (Mining Pool B) types of nodes. Note that *unreachable* nodes cannot connect with each other. Therefore, a block must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

(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 network synchronization. (4) This block relaying method is followed even when miners use fast relay networks [27]. (5) Since *reachable* nodes are the entry points for a block in the network, if a crawler connects to a large number of *reachable* nodes, it can label these entry points as mining nodes.

B. Data Collection System

We deployed a crawler in the Bitcoin network to connect with the *reachable* nodes using the RPC API. Our crawler node also accepted incoming connections from other *reachable* and *unreachable* nodes. We conducted our study for 5,750 consecutive blocks (≈47 days). The default Bitcoin Core client does not support large-scale network mapping by limiting

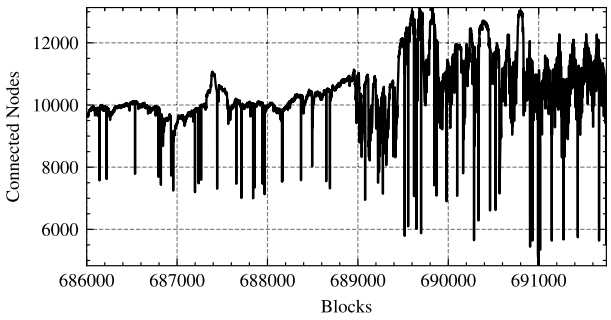


Fig. 3. The number of nodes to which our crawler connected during 47 days of data collection. The x-axis denotes the time represented as the number of consecutive blocks and the y-axis denotes the node count. We connected to ≈103K nodes during the measurement duration with ≈10,459 *reachable* nodes present in the network at any time. The maximum number of *reachable* nodes present in the network at any time was 13,134.

the outgoing connections to 10 nodes only. To overcome this limitation, we modified the Bitcoin source code and increased the file descriptor limits on our crawler. We did not make any additional changes to the source code (*i.e.,* disable block forwarding) to comply with the ethical standards.[6] In 47 days, we connected to 103,895 unique IP addresses, including 70,148 IPv4, 24,895 IPv6, and 8,852 Tor addresses. In Figure 3, we show the number of nodes to which our crawler was connected at any time. We found that on average, ≈10,459 nodes were present in the network. The maximum number of nodes to which we connected at any time was 13,134.

Compared to our previous measurements in [1], our new experiments revealed that the Bitcoin network size has increased over the years. In [1], we connected to 36,360 unique IP addresses over five weeks, including 29,477 IPv4, 6,391 IPv6, and 522 Tor addresses. In our new experiment, we connected to 103,895 unique IP addresses, including 70,148 IPv4, 24,895 IPv6, and 8,852 Tor addresses. Although there is a general increase among all address types, the number of Tor addresses increased by ≈16 times, indicating that Bitcoin users are adopting more privacy-preserving node hosting techniques. Figure 2 provides an illustration of our data collection system in the context of the Bitcoin peer-to-peer network.

IV. Identifying the Mining Nodes

To detect the mining nodes, we used the Bitcoin RPC API to sample the network information and developed a heuristic to detect the mining nodes. The Bitcoin RPC API command *getblockchaininfo* provides information about the latest block on the blockchain. 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 that invoked the *getpeerinfo* API. The *getpeerinfo* renders up-to-date interactions with all connected peers. A sample interaction with one peer is shown in Figure 4 and the key variables to note are "addr", "lastsrecv", "synced_headers", "synced_blocks",

[6]When connected with thousands of nodes in the network, disabling block forwarding may cause some concerns since connected nodes may not benefit from the opportunity of receiving a newly mined block earlier than they actually receive it.

```
{ id: 12188,
  addr: '169.x.x.x:8xxx',
  addrlocal: '132.x.x.x:8xxx',
  addrbind: '132.x.x.x:8xxx',
  lastsend: 1554493200,
  lastrecv: 1554493185,
  version: 70015,
  subver: ''/Satoshi:0.16.0/'',
  startingheight: 569534,
  synced_headers: 570367,
  synced_blocks: 570366,
  inflight: [ 570367 ], }
```

Fig. 4. 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.

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. In terms of our ideal world functionality, "synced_blocks" renders the view $\text{VIEW}_{\mathcal{C}}^{P_i}$ 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 a heuristic to detect the mining nodes.

A. Heuristic

For a peer P_i, when the blockchain view is updated from $\text{VIEW}_{\mathcal{C}}^{P_i}$ to $\text{VIEW}_{\mathcal{C}+1}^{P_i}$, 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.

The heuristic above is a mapping between the information exposed by the RPC API and the Bitcoin network traffic of a crawler. For more clarity on the heuristic, revisit Figure 4 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 4. For each peer, the crawler checks if "synced_headers" value is 570367 ($\mathcal{C}+1$). If the crawler is downloading that block from the mining node, the "inflight" value is also set to $\mathcal{C}+1$. The example in Figure 4 shows that the "inflight" value is "570367," hence the "addr" is the mining node's IP address.

B. Results

We report our findings in Figure 5. To preserve the anonymity of the mining nodes, we mask the last two octets of their IP addresses. Overall, we discovered 871 mining nodes. Among them, 637 (73.1)% used IPv4, 163 (18.7%) used IPv6, and 71 (8.15)% used Tor (OnionCat) addresses. Our results indicate that the mining pools use Tor to shield their nodes from the routing attacks [28]. Compared to our previous study conducted last year [1], our new results show a notable change in the Bitcoin P2P network. Previously,

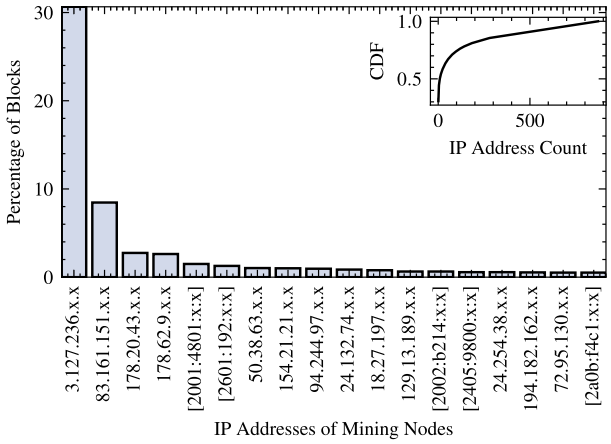


Fig. 5. Results obtained by applying the mining node detection heuristic. The histograms show the percentage of blocks contributed by mining nodes. We mask the IP addresses of the mining nodes to preserve their privacy. The subplot is the CDF showing the distribution of IP addresses with respect to the blocks produced.

we identified 359 mining nodes in the network during 35 days of data collection. Our recent results indicate an increase in the number of mining nodes with two plausible explanations. First, due to an increase in the network hash rate, more mining pools have joined the network thereby increasing the number of mining nodes. Second, a significant mining decentralization has occurred over the years with the percentage of *unknown* mining pools increasing from 10% in 2020 to $\approx$ 48% in 2021 [29]. The decentralized mining power has resulted in new miners joining the network.

With the overall increase in the network size (§III-B), including the number of mining nodes, a few logical inferences can be made regarding the Bitcoin network. First, since the number of outgoing connection slots in Bitcoin Core has not changed in the last few years, an increase in the number of nodes would have logically contributed to the network asynchrony. Second, the increase in the number of mining nodes reflects that Bitcoin mining is displaying trends of decentralization. As such, mining decentralization along with an increasing network size means a higher diversity in the block propagation patterns. As we later show in §VI, non-uniform block propagation is at the core of network asynchrony and the *HashSplit* attack. Therefore, our new results indicate that the Bitcoin network has become more vulnerable to the attack.

V. Bitcoin Network Asynchrony

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 the variance in the block propagation pattern of the mining nodes based on their network reachability. Using that observation, we validated the network asynchrony in Bitcoin.

The most commonly referenced theoretical model in this context is by Pass et al. [7] 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. [7] assumed that after Δ, all the mining nodes simultaneously receive a block to start the next round. Although Pass et al. [7] called their model "asynchronous," however, Ren [8] showed that the model is actually *non-lock-step* synchronous.

Ren [8] further specified that an asynchronous model is weaker than the *non-lock-step* synchronous model whereby an adversary can arbitrarily delay blocks in order to fork the blockchain. In other words, one of the key outcomes of the asynchronous model is a fork caused by the delay in block delivery. Note that in our ideal world functionality, if miners receive two valid blocks in a round, they will mine on the block that they receive first while keeping the other block as a fork. We observe that as long as the network is neither *lock-step* synchronous nor *non-lock-step* synchronous, such forks can be created without arbitrarily delaying the block delivery to the mining nodes. This situation can be created if the mining nodes receive at least two blocks in each round and both blocks are received at different times. In line with Ren's [8] definition, a network where this mining activity can be orchestrated is weaker than both *lock-step* synchronous and *non-lock-step* synchronous networks. Therefore, we call it an asynchronous network where heterogeneous latency combined with the principles of block mining, allows the adversary to fork the blockchain.

As mentioned above, the prerequisite to an asynchronous 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 [8]. In §VI, 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.

To analyze the block propagation pattern among the mining nodes, we collected information from Bitnodes data propagation API [30].[7] The Bitnodes data propagation API reports the block reception time for the first 1,000 *reachable* nodes that report the block before the other *reachable* nodes. For our analysis, we assumed that the 871 mining nodes are among those fast 1,000 nodes. Our assumption in this case helps in constructing a lower bound estimate for the adversary in order to partition the mining nodes. If 871 nodes with the minimum block propagation delay experience non-uniform delay in the block reception, we can generalize that observation across the actual mining nodes. Moreover, if in practice, the mining nodes are not among the 1,000 nodes with the minimum block propagation delay, then the network becomes more vulnerable to the *HashSplit* attack (§VI)

Block Propagation Among Mining Nodes: Through the data propagation API, we observed that no two block propagation patterns are the same, and some blocks propagate faster than others. In Figure 6, we show a sample from our measurements

[7]We recently discovered that the RPC API, despite being useful for mining nodes detection, may not yield desirable results for the block propagation delay in the network. For that purpose, we found an alternative source in Bitnodes data propagation API.

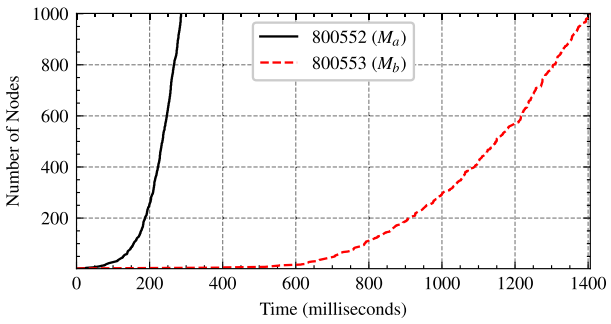


Fig. 6. Block propagation pattern of two miners M_a and M_b at heights 800552 and 800553, respectively. M_a's block reached $|M|/2$ miners in 228 milliseconds and $|M|$ miners in 276 milliseconds. M_b's block reached $|M|/2$ miners in 1,108 milliseconds and $|M|$ miners in 1,337 milliseconds.

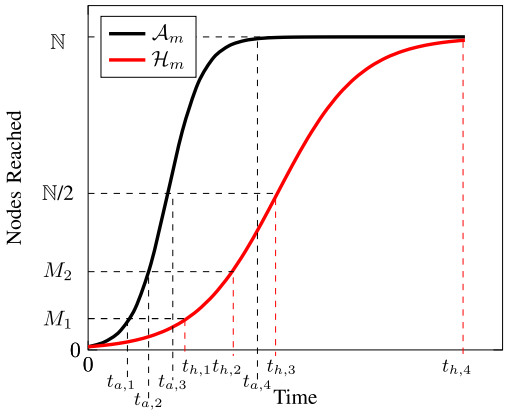


Fig. 7. Generalized illustration of Figure 6. For simplicity of modelling, we assume a uniform hashing power distribution in M (*i.e.,* $M_1 \approx |M|/2 \approx$ 51% hash rate and *i.e.,* $M_2 \approx |M|/2 \approx$ 49% hash rate). $\mathcal{A}_m$ is well connected compared to $\mathcal{H}_m$. If $\mathcal{H}_m$ 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, $t_{h,1} \ldots t_{h,4}$ are the corresponding values for $\mathcal{H}_m$.

consisting of two consecutive blocks at heights 800552 and 800553, respectively. For simplicity, we refer to the miners of blocks 800552 and 800553 as M_b and M_a, respectively. We observed that M_a's block propagated faster than block M_b's block. M_a's block reached 435 nodes in 228 milliseconds, and 871 nodes in 276 milliseconds. In contrast, M_b's block reached 435 nodes in 1,108 milliseconds and 871 nodes in 1,337 milliseconds. As discussed in the previous section, we use $|M| = 871$ nodes with the minimum propagation delay as an illustrative example to model the mining nodes. Therefore, $|M|/2 = 435$ nodes become 50% of the mining nodes.

From the analysis above, we derived the following conclusions. (1) The current Bitcoin network is neither *lock-step* synchronous [6] nor *non-lock-step* synchronous [7], [31], [32]. (2) The block propagation pattern suggests that the mining nodes do not form a completely connected $M \times M$ topology.[8] 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 varying network reachability. (4) Based on the observed network characteristics and the specifications provided in [8], the Bitcoin network can be considered asynchronous.

VI. 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 [7], [11], [13]. Moreover, as indicated by Pass et al. [7], the problem becomes worse if the network is fully asynchronous, thus allowing an adversary to mount new attack strategies 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 and it shows that an adversary with 26% hash rate can violate both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ with high probability. The key idea behind *HashSplit* attack is that due to varying block propagation pattern, an adversary can compute a block and withhold it until another honest miner also produces a block. As the honest miner's block propagates in the network, the adversary releases its own block to 51% of the mining nodes. As mentioned in the ideal world functionality, when a miner receives more than one valid block in a round, it mines on the earliest received block. As a result, the 51% of the miners mine on the adversary's block while 49% of the miners mine on the other block. With 51% of the miners working to extend the adversary's block, the probability of the honest miner's block being accepted in the long run reduces in each round. By adopting this strategy, the adversary can continuously fork the chain whenever it mines the block and keep invalidating the work of other honest miners, thereby violating the chain quality. In case the adversary sustains the fork for a few blocks, it may also violate the common prefix property.

A. Threat Model and Attack Objectives

For *HashSplit*, we assume an adversary $\mathcal{A}_m \in M$ with less than 51% hash rate. $\mathcal{A}_m$ follows the experiment methodology in §III–§V to connect to all $P_i \in \mathbb{N}$, identify the mining nodes M, and estimate their hashing power using the block mining rate (Figure 5). Additionally, $\mathcal{A}_m$ follows the measurement technique of Bitnodes data propagation API [30] to obtain the block propagation pattern of each mining node in the network. Using the block propagation pattern, $\mathcal{A}_m$ calculates how a block generated by each $P_i \in M$ reaches all $P_i \in \mathbb{N}$. $\mathcal{A}_m$ then maintains a direct connection with each mining node to instantly send or receive blocks. If $\mathcal{A}_m$ samples the block

[8]Prior work [10] 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.

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 patterns 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. From Figure 6, $\mathcal{A}_m$ can precisely determine the time at which 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 [5] and is currently deployed in Bitcoin Core clients. Finally, we assume that (1) $\mathcal{A}_m$ cannot influence the communication model of other $P_i \in \mathbb{N}$ by launching routing attacks [9], [15], 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 [7], [9], [11].

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 [5]. 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.

B. Attack Procedure

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

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

$P_i \in M$, 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 $\delta_1 \ldots \delta_4$ 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 $\delta_1 = 880$, $\delta_2 = 1061$ milliseconds, 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. Note that we have used four discreet time windows ($\delta_1 \ldots \delta_4$) for simplicity. In practice, the adversary can compute the mining node's reachability with respect to any *reachable* mining or non-mining node as shown in Figure 6. This computation will increase the attack precision.

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 [5]). 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), $\mathcal{A}_m$

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.

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) 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 §VI-A, the perfect split leads to $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$. If $\mathcal{A}_m$, with 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.

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 [33]. 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 [33]. 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 $t'_o/(t_o + t'_o) = \alpha/(\alpha+\beta)$ [25], [33], [34]. 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 get resolved before k blocks and $\mathcal{C}_1$ wins, $\mathcal{Q}_{cq}$ is violated, and the attack succeeds partially.
2) 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.
3) 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 **F** and **R** as the sum of events in which forks persist or get resolved. In (5) and (6), we show the probability $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$.

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

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

$$- \beta_2 - \alpha_2 - \beta_1 + 1 \tag{5}$$

$$\mathbb{P}[X = \mathbf{R}] = \alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) + \beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1)$$

$$\mathbb{P}[X = \mathbf{R}] = 2\alpha_1\alpha_2\beta_1 + 2\alpha_1\alpha_2\beta_2 + 2\alpha_1\beta_1\beta_2 - 3\alpha_1\alpha_2\beta_1\beta_2 + 3\alpha_2\beta_1\beta_2 - \alpha_1\alpha_2 - \alpha_1\beta_1 - \alpha_1\beta_2 - 2\alpha_2\beta_1 - 2\alpha_2\beta_2 - 2\beta_1\beta_2 + \alpha_2 + \beta_1 + \beta_2 \tag{6}$$

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. leftmargin=0.5cm

1) With algorithm 2 as the starting point of a block race, there is a higher probability that the given fork persists or new forks appear. This favors the violation of $\mathcal{Q}_{cp}$.
2) The probability that a fork is resolved by an honest miner on $\mathcal{C}_1$ is $\alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) = 0.1275$; significantly less than 0.6892 and favors $\mathcal{Q}_{cq}$'s violation.[9]
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.
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 $\mathbb{P}[X_k = \mathbf{R}]$ [22], [35]. 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 $\mathcal{Q}_{cq}$.

a) 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 delay and the hash rate can be obtained as:

$$\alpha_1 = \frac{\tau}{t_{a,0}}, \alpha_2 = \frac{\tau}{t_{h_1,0}}, \quad \beta_1 = \frac{\tau}{t_{h_2,0}}, \beta_2 = \frac{\tau}{t_{h_3,0}} \tag{7}$$

[9]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).

$$\alpha_1 = \frac{\tau}{t_{a,0} + t_{a,1}}, \quad \alpha_2 = \frac{\tau}{t_{h_1,0} + t_{h,1}}, \quad \beta_1 = \frac{\tau}{t_{h_2,0} + t_{h,1}}, \quad \beta_2 = \frac{\tau}{t_{h_3,0} + t_{h,1}} \tag{8}$$

Considering $\alpha_1 = 0.26$, $\alpha_2 = 0.25$, $\beta_1 = 0.25$, $\beta_2 = 0.24$, and $\tau = 600$ seconds, from (7), $t_{a,0}$, $t_{h_1,0}$, $t_{h_2,0}$ become $\approx$ 2308, 2400, 2400, and 2500, respectively. Plugging these values in (8), the hash rate of each miner becomes $\alpha_1 = 0.2599$, $\alpha_2 = 0.2499$, $\beta_1 = 0.2499$, and $\beta_2 = 0.2399$. Note that $\mathcal{A}_m$ also has an additional head start mining advantage if it is the first miner to produce a block on $\mathcal{C}_1$. In Figure 8 ($\mathbf{f_1}$), when $\mathcal{A}_m$ mines the block before any other miner, it does not stop its computations. Instead, $\mathcal{A}_m$ continues to mine the next block until another miner produces a block on either $\mathcal{C}_1$ or $\mathcal{C}_2$, thereby maximizing $\mathcal{A}_m$'s advantage over other miners. In case $\mathcal{A}_m$ succeeds in mining the subsequent block, it follows algorithm 2 to maintain the perfect split by only releasing the block that is at the same height as the competitive block. Although this mining advantage can be characterized to gain deeper insights into the block race, it might add more complexity to our current analysis. Therefore, for simplicity, we do not consider the head start mining advantage and instead rely solely on the propagation delay advantage. By following this approach, we recognize that the attack success probability might be less than the true probability that can be obtained by incorporating the head start mining advantage. However, even with a potentially lower success rate, we can still show that $\mathcal{A}_m$ can violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ in the block race.

Using $\alpha_1 = 0.2599$, $\alpha_2 = 0.2499$, $\beta_1 = 0.2499$, and $\beta_2 = 0.2399$, $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$ become 0.6892 and 0.3108, respectively. Using these values, we can construct the transition probability matrix for Figure 9.

$$P = \begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \left\| \begin{array}{cc} p_{00} & p_{01} \\ p_{10} & p_{11} \end{array} \right\| \end{array} = \begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \left\| \begin{array}{cc} 0.6892 & 0.3108 \\ 0.6892 & 0.3108 \end{array} \right\| \end{array}$$

In (9), 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 $\pi P = \pi$.

$$\begin{aligned} 0.6892\pi_1 + 0.3108\pi_2 &= \pi_1 \\ 0.6892\pi_1 + 0.3108\pi_2 &= \pi_2, \quad \pi_1 + \pi_2 = 1 \end{aligned} \tag{9}$$

From (9), $\pi_1 = 0.689$ and $\pi_2 = 0.311$, and the long-term probability of a forked chain is greater than the probability of a single branch. Using the stationary distribution, we evaluate the impact of *HashSplit* on $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

b) Common prefix property: Our analysis reveals that for any block race of length k, $\mathcal{Q}_{cp}$ is violated ($\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$) with 0.69 probability. For $k = 6$, P^6 yields $\mathbb{P}[X = F] = 0.69$. Therefore, *HashSplit* violates $\mathcal{Q}_{cp}$ with a high probability.

c) 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 0.2053. We analyze that by branching S_1 in Figure 9 into two states and calculate the probability of $\mathcal{C}_2$ being the winning

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 $\beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2)+\beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1)$.

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

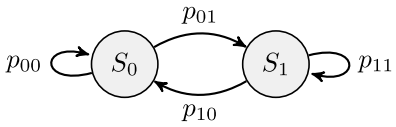


Fig. 9. State machine representation of a block race. Transition probabilities are p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X=\mathbf{F}]$, $\mathbb{P}[X=\mathbf{R}]$, $\mathbb{P}[X=\mathbf{F}]$, and $\mathbb{P}[X=\mathbf{R}]$, respectively.

chain (computed as 0.21). Therefore, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated with a probability of $1-0.21=0.79$.[10]

In summary, *HashSplit* violates the blockchain *safety* and chain quality with a high probability. Moreover, our attack construction in §VI-B is modeled on nodes with the minimum block propagation delay. In practice, if the mining nodes are not among the nodes with the minimum block propagation delay, the attack becomes more feasible. As noted in §I, splitting the mining power to lower the cost of the 51% attack is known in the literature [9], [15], [16]. 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 model. Instead, the notable aspect of *HashSplit* is the evaluation of the network behavior when mining nodes receive blocks at different times, and if presented with two valid blocks, they chose to mine on the block that they receive first. These two aspects create an opportunity for an adversary to leverage the network asynchrony and fork the chain when the adversary mines a new block. These characteristics highlight the novelty of *HashSplit* over the other attacks proposed on Bitcoin.

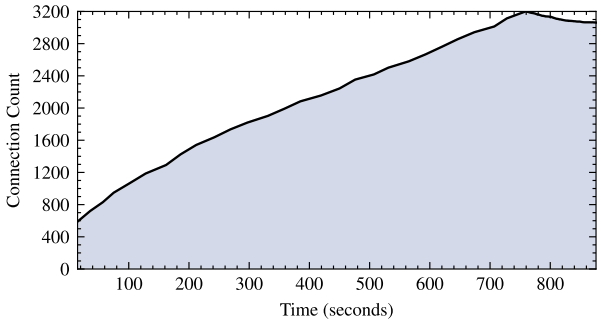
[10]Another way to analyze $\mathcal{Q}_{cq}$ violation is by taking into account $\mathcal{A}_m$'s effective hash rate which is more than its default hash rate due to higher block propagation speed. Moreover, since $\mathcal{A}_m$ frequently forks the chain after mining blocks, it naturally increases the number of orphaned blocks. As a result, if a branch containing $\mathcal{A}_m$'s blocks wins, honest miners do not mine blocks proportional to their hash rate which violates $\mathcal{Q}_{cq}$.

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 can significantly vary with time (see [30]). However, irrespective of those changes, as long as the block propagation pattern of the mining nodes is different from each other, *HashSplit* can be launched by an adversary with better network reachability.

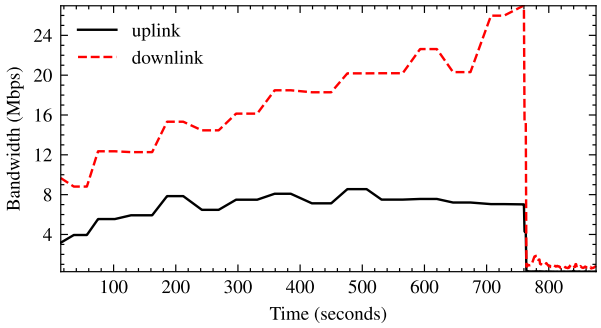
VII. Attack Countermeasures

In this section, we discuss the attack countermeasures. It can be argued that mining pools can prevent the *HashSplit* attack by using dedicated relay servers to communicate blocks. However, we note that using such techniques have some key challenges that may inhibit their adoption. First, blockchain systems that use Nakamoto consensus are considered to be decentralized, with no apriori coordination among entities that aim to mine blocks. Second, since the blockchain peer-to-peer networks are *permissionless*, new miners can join the network at any time and use the default network to relay their blocks. As such, the new miners may be at a disadvantage of using the default network to relay blocks if other miners are using a faster relay network. Keeping these challenges in mind, a more practical approach to counter *HashSplit* is by to enhancing the connectivity among the mining nodes in the peer-to-peer network as well as adding application layer defenses. Our proposed countermeasures below showcase the possibilities of creating such countermeasures in Bitcoin.

Since *HashSplit* exploits asynchronous network and block propagation pattern, if $\delta_1 \ldots \delta_4$ in Algorithm 1 yield to 0, $\mathcal{A}_m$ cannot split the mining nodes or leverage a significant mining advantage [36]. Additionally, M can form $M \times M$ topology,



(a) Number of Bitcoin connections established in 1800 seconds



(b) Incoming and outgoing bandwidth consumption in Mbps.

Fig. 10. Evaluation of our Bitcoin client deployed on a custom node. In ≈700 seconds, the node connected with over 3K reachable nodes. The average bandwidth consumption over the experiment duration (876 seconds) was under 8MBPs. The bandwidth consumption was high during the initial connectivity phase after which it reduced as the connections stabilized.

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 the local version of the Bitcoin Core client [19]. 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 good network reachability to form $M \times M$ topology. For instance, it can take several days for all incoming connection slots to be full [37]. If those incoming connections include 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 our Bitcoin Core client by adding scripts that allow faster connectivity. We increased the file descriptor limits on the local machine, crawled IP addresses of *reachable* nodes from Bitnodes [30], and used Bitcoin RPC API to connect with the *reachable* nodes. To expedite the connectivity process, we used a shell tool to enable multiple connection requests in parallel [38].

For performance evaluation, we deployed our client on a DigitalOcean droplet with 16GB memory and 8 CPUs. Previously, we conducted the experiment on a local lab machine with better specifications. However, since most Bitcoin nodes operate on cloud [30], we re-conducted the experiment on a cloud machine so that our results are more generalizable. In our experiment, we evaluated the connectivity speedup and bandwidth consumption, with results reported in Figure 10. Our node connected with over $3K$ reachable nodes in ≈700 seconds, with the average bandwidth consumption under 8MBps (1.9MBps incoming and 5.3Mbps outgoing). The bandwidth consumption was high during the initial connectivity phase, after which it dropped significantly as the connections stabilized. The results reported in Figure 10 were taken from an experiment duration of 876 seconds while only connecting to IPv4 and IPv6 nodes.

From Figure 10, we note that by direct connectivity and better reachability, the node can instantly receive blocks from honest mining nodes, thereby minimizing $\mathcal{A}_m$'s advantage. Moreover, by leveraging publicly available data of Bitnodes and establishing parallel connections through RPC API, new mining nodes can quickly connect with a large number of reachable nodes (including the mining nodes). During the process, the overall bandwidth consumption is tolerable, especially after connections stabilize.

We acknowledge that even after connecting to all mining nodes through this process and ideally forming $M \times M$ topology, the risk of *HashSplit* attack may still exist. Due to characteristics of the underlying Internet infrastructure (*i.e.,* low bandwidth), the network latency can be heterogeneous such that two peers connected to the 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 *HashSplit* attack.

For application layer defenses, we applied a *fork resolution* mechanism in our local Bitcoin Core client [19]. 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 leveraged this sequence of block arrival to eliminate $\mathcal{A}_m$'s advantage and reduce the likelihood of a perfect split. In our proposed *fork resolution* method, a node removes the connection and bans the IP address for twenty-four hours in the event of receiving $k = 6$ consecutive blocks from it [19]. This means that $\mathcal{A}_m$ loses 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 the victim node. Therefore, a combination of high network reachability and *fork resolution* mechanism can alleviate the risk of the *HashSplit* attack.

VIII. 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] M. Saad, A. Anwar, S. Ravi, and D. Mohaisen, "Revisiting Nakamoto consensus in asynchronous networks: A comprehensive analysis of Bitcoin safety and ChainQuality," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 988–1005.

[2] P. Das et al., "FastKitten: Practical smart contracts on Bitcoin," in *Proc. USENIX Secur. Symp.*, N. Heninger and P. Traynor, Eds., 2019, pp. 801–818. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/das

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

[4] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for Bitcoin," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 817–831, doi: 10.1145/3319535.3354237.

[5] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] J. A. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin backbone protocol with chains of variable difficulty," in *Advances in Cryptology* Cham, Switzerland: Springer, 2017, pp. 291–323, doi: 10.1007/978-3-319-63688-7_10.

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

[8] L. Ren, "Analysis of Nakamoto consensus," IACR Cryptol. ePrint Arch., Tech. Rep. 2019/943, 2019. [Online]. Available: https://eprint.iacr.org/2019/943

[9] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 375–392, doi: 10.1109/SP.2017.29.

[10] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering Bitcoin's public topology and influential nodes," Univ. Maryland, College Park, MD, USA, Tech. Rep., 2015, p. 54. [Online]. Available: https://www.cs.umd.edu/projects/coinscope/coinscope.pdf

[11] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *Proc. IEEE P2P*, Sep. 2013, pp. 1–10, doi: 10.1109/P2P.2013.6688704.

[12] R. Zhang and B. Preneel, "Lay down the common metrics: Evaluating proof-of-work consensus protocols' security," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 175–192, doi: 10.1109/SP.2019.00086.

[13] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on Bitcoin: Colliding space, time, and logic," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1175–1187, doi: 10.1109/ICDCS.2019.00119.

[14] A. E. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. G. Sirer, "Decentralization in Bitcoin and Ethereum networks," 2018, *arXiv:1801.03998*.

[15] C. Natoli and V. Gramoli, "The balance attack or why forkable blockchains are ill-suited for consortium," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 579–590, doi: 10.1109/DSN.2017.44.

[16] 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 *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 3–16, doi: 10.1145/2976749.2978341.

[17] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. USENIX Secur. Symp.*, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[18] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS¶)*, Mar. 2016, pp. 305–320, doi: 10.1109/EuroSP.2016.32.

[19] Anonymous. (2020). *Improved Bitcoin Core to Counter Hashsplit*. [Online]. Available: https://anonymous.4open.science/r/56e77487-0470-4e10-b634-b13e939863c0/

[20] C. Wang, X. Chu, and Q. Yang, "Measurement and analysis of the Bitcoin networks: A view from mining pools," 2019, *arXiv:1902.07549*.

[21] B. Community. (2019). *Six Confirmation Practice in Bitcoin*. [Online]. Available: https://en.bitcoin.it/wiki/Confirmation

[22] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*. Cham, Switzerland: Springer, 2014, pp. 436–454, doi: 10.1007/978-3-662-45472-5_28.

[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 *Proc. Conf. Comput. Commun. Secur.*, 2017, pp. 195–209, doi: 10.1145/3133956.3134019.

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

[25] C. Grunspan and R. Pérez-Marco, "On profitability of selfish mining," 2018, *arXiv:1805.08281*.

[26] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in Bitcoin," in *Financial Cryptography Data Security*. Cham, Switzerland: Springer, 2016, pp. 515–532, doi: 10.1007/978-3-662-54970-4_30.

[27] R. Nagayama, R. Banno, and K. Shudo, "Identifying impacts of protocol and internet development on the Bitcoin network," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–6, doi: 10.1109/ISCC50000.2020.9219639.

[28] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: Protecting Bitcoin against routing attacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/

[29] BitcoinCommunity. *Bitcoin Hash Rate Distribution*. [Online]. Available: https://www.blockchain.com/charts/pools

[30] BitcoinCommunity. (2020). *Bitnodes: Discovering All Reachable Nodes in Bitcoin*. [Online]. Available: https://bitnodes.earn.com/

[31] J. Zhao, J. Tang, Z. Li, H. Wang, K.-Y. Lam, and K. Xue, "An analysis of blockchain consistency in asynchronous networks: Deriving a neat bound," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 179–189, doi: 10.1109/ICDCS47774.2020.00039.

[32] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 729–744, doi: 10.1145/3243734.3243814.

[33] C. Grunspan and R. Pérez-Marco, "Double spend races," 2017, *arXiv:1702.02867*.

[34] M. Rosenfeld, "Analysis of hashrate-based double spending," 2014, *arXiv:1402.2009*.

[35] Q.-L. Li, Y.-X. Chang, X. Wu, and G. Zhang, "A new theoretical framework of pyramid Markov processes for blockchain selfish mining," 2020, *arXiv:2007.01459*.

[36] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "Modeling the impact of network connectivity on consensus security of proof-of-work blockchain," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1648–1657, doi: 10.1109/INFOCOM41043.2020.9155451.

[37] 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 *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 894–909, doi: 10.1109/SP40000.2020.00027.

[38] O. Tange et al., "GNU parallel-the command-line power tool," *USENIX Mag.*, vol. 36, no. 1, pp. 42–47, 2011.