2021

Analyzing the Blockchain Attack Surface: A Top-down Approach

Muhammad Saad
University of Central Florida

ANALYZING THE BLOCKCHAIN ATTACK SURFACE: A TOP-DOWN APPROACH

by

MUHAMMAD SAAD
B.E. Electrical Engineering, National University of Science and Technology, Pakistan, 2014
M.S. Electrical Engineering, Lahore University of Management Sciences, Pakistan, 2017

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2021

Major Professor: David Mohaisen

ABSTRACT

Blockchains enable secure asset exchange in a distributed system, thereby facilitating innovative applications such as cryptocurrencies and smart contracts. Although the cryptographic constructs of blockchains are highly secure, however, their practical deployments are vulnerable to various attacks due to their application-specific policies, and their peer-to-peer (P2P) network intricacies. In this work, we take a top-down approach towards exploring those attacks, starting with the application-specific abuse of blockchain-based cryptocurrencies and concluding with the network conditions that violate the blockchain consistency.

In the top-down approach, we first analyze the application-specific abuse of blockchain-based cryptocurrencies by uncovering (1) covert cryptocurrency mining in the web browsers, and (2) artificially inflating the transaction fee by attacking the blockchain memory pools. For both attacks, we show how the application policies are exploited to affect the benign users.

After exploring the application-specific attacks, we proceed towards a systematic analysis of inconsistencies in the blockchain P2P network. For this analysis, we focus on Bitcoin which is the most dominant blockchain system. Our analysis reveals that the biased distribution of resources in the Bitcoin network can be exploited to launch various partitioning attacks. Furthermore, through a root cause analysis, we discover that (1) the Bitcoin network is asynchronous in the real world, and (2) its security model does not embrace the risks associated with network churn.

The last two components in the dissertation consolidate our attack surface analysis by analyzing the impact of network asynchrony and network churn on the blockchain consistency property. We conduct theoretical analysis and measurements to show how various network characteristics can be exploited to reduce the cost of launching notable attacks that violate consistency.

Our top-down approach uncovers various novel attacks that have not been studied in the prior works. For each attack, we also propose countermeasures to harden the blockchain security.

To my grandparents Abdul Aziz, Khurshid Bibi, Muhammad Saleem, Qayyum Akhtar, and Shareefan Bibi

ACKNOWLEDGMENTS

This work would not have been possible without the support of many individuals to whom I am grateful. Foremost, I am thankful to my parents (Shagufta and Zulfiqar) and my siblings (Ameer, Nida, and Zaid) for their continuous support and love.

I would like to thank my advisor and doctoral committee chair, Dr. David Mohaisen, for being my inspiration and helping me grow as a researcher. I am extremely proud to be your student. I also thank my dissertation committee members for their support: Dr. Murat Yuksel, Dr. Sung Choi Yoo, and Dr. Changchun Zou.

Gratitude to my collaborators for their valuable contributions and feedback: Charles Kamhoua (US Army Research Lab), DaeHun Nyang (Ewha Womans University), Laurent Njilla (US. Air Force Research Lab), Mihai Christodorescu (Visa Research), My T. Thai (University of Florida), Murat Yuksel (University of Central Florida), Mahdi Zamani (Visa Research), Ranjit Kumaresan (Visa Research), Songqin Chen (George Mason University), Srivatsan Ravi (University of Southern California), and Sachin Shetty (Old Dominion University). I am also grateful to my team members at SEAL, particularly Afsah, Ahmed, Ashar, Hisham, Jeman, Mo, and Rhongho for their support.

My research career was initially nurtured by Dr. Fareed Zaffar, Dr. Ijaz Naqvi, Dr. Tariq Jadoon, Dr. Zartash Uzmi, Dr. Ihsan Qazi, Saloo Durrani, and Ahmad Javed. You are still my guiding light. I am also grateful to my cousin Dr. Zubair Shafiq for being my strength in challenging times. Gratitude to my friends Abdullah, Amin, Aubrey, Aysha, Cameron, Emily, Imran, Jafar, Mehboob, Mervat, Mohsin, Olga, Shahzaib, Seher, Shehzad, Soha, Tulha, and Zainab for their care.

Finally, I want to thank my girlfriend Mehar Khan who stood by my side through thick and thin. You are the most important “Related Work” in my life.

This work was supported by the Air Force Material Command under FA8750-16-03011, the National Research Foundation under Grant NRF-2016K1A1A2912757 and NSF under Grant CNS-1809000. All opinions in this work are of the authors and do not reflect those of the sponsors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Blockchain technology has significantly advanced the field of distributed systems by enabling novel applications including cryptocurrencies and smart contracts. A blockchain system is actuated by nodes that exchange transactions in network and those transactions are recorded in an append-only blockchain ledger. To maintain *consistency*, nodes execute a consensus protocol that specifies the rules of a *valid* blockchain. In blockchain systems, Proof-of-Work (PoW) is the most prevalent consensus protocol and it guarantees consistency if an honest majority possesses more than 50% mining power required to mine blocks.[1] The blockchain security is provisioned by the strong cryptographic constructs of the blockchain data structure and the consensus protocol.

However, blockchain security does not completely rely on strong cryptographic primitives alone. In fact, blockchain security also depends upon application-specific design choices and various P2P network intricacies. For instance, Bitcoin restricts the block size at 1MB and the average inter-arrival time between two blocks at 10 minutes. Moreover, Bitcoin assumes a *lock-step* synchronous overlay network in which a block is concurrently relayed to all the network nodes, incurring negligible delay. As such, the security assumptions about the honest majority are only valid as long as the Bitcoin network conforms to the aforementioned rules. Logically, a deviation from those rules creates an attack surface that can be exploited to harm the system.

Moreover, attacks related the blockchain applications are not limited to the blockchain system itself. In practice, blockchain systems can also be used as a standalone attack vector to affect other critical systems such as the cloud infrastructure. For example, PoW-based cryptocurrencies such as Bitcoin, Ethereum, and Monero require resource-intensive hashing (also called mining) to compute a valid block. Since a valid block is rewarded with money, therefore, it motivates attackers to compromise the capable infrastructure and covertly mine those blocks. Although such activities do not affect the blockchain systems in principle, they occur due to blockchain application policies,

[1] In PoW-based blockchain systems, mining power is also called the hash rate possessed by the miner.

thereby causing a general security concern in the community.

Given these security concerns, the blockchain attack surface has various components associated with the application-specific design choices and the P2P network dynamics. We categorize them as the application-specific attacks and the network layer attacks. To elaborate on each attack category, consider the Bitcoin blockchain that specifies a block size limit of 1MB. To launch an application-specific attack, an adversary can exploit the block size limit to flood blocks with dust transactions and delay the confirmation of other high-value transactions [7]. The attack is feasible in applications that specify a smaller block size limit (*i.e.* 1MB in Bitcoin compared to 8MB in Bitcoin Cash). An application-specific attack example can be found in [7], which shows that Bitcoin blocks have been frequently targeted with dust transactions.

To launch a network layer attack, an adversary can exploit the physical network characteristics (*i.e.* latency) to prevent a group of nodes from timely receiving a block. In cryptocurrencies, block propagation delay leads to the mining power reduction [34]. An adversary can delay the block propagation by controlling either the overlay topology among the blockchain nodes [55], or the physical network resources (*i.e.* BGP routers [5]). A network layer attack example can be found in [5], which shows that by exploiting the biased distribution of Bitcoin nodes across Autonomous Systems (ASes), an adversary can reduce the network's mining power by more than 60%.

In keeping with the distinct nature of each attack category, in this dissertation, we take a top-down approach towards the blockchain attack surface analysis, starting with the application-specific attacks including cryptojacking and memory pool denial-of-service attack. We then proceed towards the network layer attacks including partitioning attacks resulting from the inconsistencies in the Bitcoin network. Finally, we consolidate our attack surface analysis by combining the application-specific policies with the network layer inconsistencies to present novel attacks that feasibly violate the fundamental blockchain properties.

1.1 Motivation

In the above, we elaborated on various concepts in the advance of blockchain security that require a comprehensive attack surface analysis to expose the vulnerabilities in the blockchain systems. Towards that goal, this dissertation is of significant importance for the following reasons.

First, blockchain-based cryptocurrencies have a high financial value which makes them lucrative attack avenues for financial gains. Currently, the market capitalization of all cryptocurrencies is ≈$1.5 Trillion [16]. Second, notable companies (*i.e.* Paypal) are now provisioning cryptocurrency services to their users by enabling cryptocurrency trading on their legacy platforms. Given the increasing adoption of cryptocurrencies and their high financial value, we foresee an increase in the attacks on those cryptocurrencies that can result in significant financial losses. Foreseeing and preventing such attacks is significantly important since cryptocurrencies are decentralized and pseudonymous by design. Therefore, if an attack is launched, the resulting damages cannot be easily recovered. A recent example can be found in the infamous Mt. Gox incident where attackers stole $460 Million worth of bitcoins from the cryptocurrency exchange [42].

In the wake of such threats, this dissertation uncovers various novel attacks that can be feasibly launched to target blockchain systems or other legacy systems. In particular, we highlight that the current blockchain systems are not designed with the security-first approach, making them an easier target due to application-specific policies or network layer inconsistencies. For example, in 2017, a group of attackers exploited the gap between the Bitcoin block size and the memory pool size to stop the confirmation of 115K transactions worth more than $700 Million [73]. Since there was no memory pool policy in effect to prevent the attack, it was frequently launched on the Bitcoin users to prevent transaction confirmation and increase the transaction fee.

Similarly, if we survey the cryptocurrency network intricacies, we find that the Nakamoto consensus only provides strong security guarantees in a *lock-step* synchronous network which assumes a completely connected topology. Therefore, the real world blockchain systems must follow a *lock-step* synchronous network to meet the required security specifications. However, no prior study

has empirically determined if the current networks follow those specifications. Our measurements reveal that the Bitcoin network is asynchronous in practice, thus incapable of meeting the security requirements. An effect of asynchrony has been recently observed in Bitcoin where a fork invalidated $319K worth of Bitcoin transactions [15].

Another research gap in the literature is that the application-specific vulnerabilities and the network layer inconsistencies have not been jointly analyzed to fully characterize the blockchain attack surface. As a result, the existing attack models only preset limited attack strategies, often requiring strong adversaries that control a significant mining power or a large number of IP addresses. In contrast, our joint analysis of the two attack categories reveals interesting insights, allowing an adversary to mount various new attack strategies to violate the blockchain consistency. We show that by acquiring only 28 IP addresses with a total cost of under $1000, an adversary can paralyze the entire Bitcoin network and double-spend without using any mining power.

In summary, our motivation is to build on the research gaps in prior works and draw attention to various security vulnerabilities in blockchain systems. Predominantly, we take a data-driven approach to empirically demonstrate that the current blockchain systems do not fully meet the security requirements in practice. Therefore, with the expansion of the cryptocurrency ecosystem, there is an imperative need to put security as a preference. Through this dissertation, we contribute towards that goal by providing a comprehensive overview of the blockchain attack surface.

1.2 Research Statement and Dissertation Organization

Based on the research gaps mentioned in the §1.1, we take a top-down approach towards the attack surface analysis. In Chapter 3 and Chapter 4, we cover the application-specific attacks, while in Chapter 5–Chapter 8, we discuss the network layer attacks. In the following, we succinctly summarize the research problems identified in each chapter along techniques used to address them.

1.2.1 Application-Specific Attacks

In the application-specific attacks, we study how the blockchain application policies can be abused to affect benign users within and outside the blockchain system. We start by analyzing in-browser cryptojacking that hijacks the resources of website users to covertly mine cryptocurrencies.

In-browser Cryptojacking (Chapter 3). In 2017, a new attack emerged in the web ecosystem in which adversaries embedded *JavaScript* codes in websites to hijack the processing power of website visitors and covertly mine PoW-based cryptocurrencies. Particularly, the adversaries selected blockchain applications that had a lower PoW *target* threshold in order to feasibly mine them by using resources of website users (*i.e.* commodity computers or smart phones). Since in-browser cryptojacking was a new attack in 2017, therefore, there was a limited understanding about its operations and impacts. To address the research gap, we collected a dataset of more than 5,700 cryptojacking websites and conducted static and dynamic analyses to study various characteristics of in-browser cryptojacking and its impact on users. Our static analysis unveiled unique code complexity characteristics of cryptojacking scripts which we then used to train machine learning models for detection. Our dynamic analysis revealed the negative impacts of cryptojacking on devices that visited cryptojacking infected websites. By further capitalizing on our dynamic analysis, we presented simple and effective methods to counter cryptojacking.

Mempool DDoS Attacks (Chapter 4). In the same year (2017), we observed an increase in the Bitcoin transaction fee and a delay in the transaction confirmation time. By taking a closer look at the Bitcoin blockchain, we found a parallel increase in the Bitcoin memory pool (also called mempool) size which stores the unconfirmed transactions. After conducting initial measurements, we discovered a high correlation between the mempool size and the transaction fee. We noticed that malicious users can exploit the gap between the block size and the mempool size to spam the mempool with dust transactions, resulting in a high transaction fee and an increased confirmation time. Capitalizing on our observations and preliminary analysis, we formally analyzed the mempool DDoS attack and proposed fee-based and age-based countermeasures that limit the attacker's

strategies without affecting benign users.

The main difference between Chapter 3 and Chapter 4 is that Chapter 3 shows how the application-specific policy (*i.e.* PoW consensus protocol) can be exploited to attack users in the web ecosystem, while Chapter 3 shows how the application policies (*i.e.* block size limit) can be exploited to target users within the blockchain system. The clear similarity between both chapters is the study of application-specific policies that can be abused to affect benign users.

1.2.2 Network Layer Attacks

In the network layer attacks, we take a data-driven approach to study the irregularities in the blockchain P2P networks and use them to construct novel and feasible attacks. Following the motivation in §1.1, our network layer analysis focuses on the problems that reveal gaps between theoretical constructs of blockchain networks and their deployments in practice.

Bitcoin Partitioning Attacks (Chapter 5). In Chapter 5, we conduct a measurement study to map the Bitcoin overlay topology on the physical network. Our results revealed that the Bitcoin nodes are highly centralized across ASes, making them vulnerable to BGP hijacks. Moreover, contrary to a common assumption that the Bitcoin network can feasibly scale up to thousands of nodes [80], our measurements revealed that the increasing network size also increases the block propagation delay which then leads to weak network synchronization. To demonstrate the effect of weak synchronization, we proposed and simulated the temporal partitioning attack which allows malicious miners to subvert nodes that experience weak synchronization.

Root Cause Analysis (Chapter 6). After observing weak synchronization and the lack of it being reported in prior works, we conducted a root cause analysis to uncover the hidden network intricacies that influence network synchronization. Continuing our data-driven approach, we explored the impact of the *unreachable* nodes, the message relaying protocols, and network churn on the Bitcoin network synchronization. Our analysis exposed several weaknesses in the current Bitcoin network which can be exploited to deteriorate network synchronization and optimize the partition-

ing attacks. The root cause analysis also revealed two novel insights about the Bitcoin network that were not observed previously. First, we observed that the Bitcoin network is asynchronous in practice, and therefore, incapable of meeting the desirable security specifications outlined by Nakamoto [76]. Second, we found that Bitcoin does not incorporate network synchronization in the security model, thus ignoring the security risks associated with it.

Exploiting Bitcoin Network Asynchrony (Chapter 7). After empirically studying the network asynchrony in Chapter 6, we combined it with application-specific mining policies to present novel network layer attack called *HashSplit*. Towards that, we first formulated the Bitcoin ideal functionality which distinctly characterized the behavior of mining and non-mining nodes. We then deployed crawlers in the Bitcoin network to identify the mining nodes and analyze blockchain propagation among them. We discovered that miners give time-based precedence to blocks that they choose to mine. Using that knowledge, we proposed the *HashSplit* attack which allows an adversary to orchestrate concurrent mining on two branches of the public chain and violate the blockchain *consistency* with a high probability. *HashSplit* is the first attack in this dissertation that combines vulnerabilities exposed by application policies and network inconsistencies.

Exploiting Network Synchronization (Chapter 8). The last chapter in this dissertation concludes the attack surface analysis by presenting the SyncAttack, an attack that exploits network churn to deteriorate synchronization and violate the blockchain consistency. For the SyncAttack construction, we incorporated network synchronization in the Bitcoin security model and conducted measurements to analyze its robustness in practice. Our analysis revealed partitioning attack possibilities created by the network churn. Additionally, by examining the Bitcoin source code, we found a major vulnerability in Bitcoin application that allows an adversary to violate network synchronization and launch a double-spend attack with a cost of $\approx$\$1000.

To summarize, in light of our motivation and objectives in §1.1, we identify problems related to the application-specific policies and the network layer inconsistencies in blockchain systems. For each set of problems, we use theoretical analysis and measurement techniques to examine their

security implications in adversarial settings. Furthermore, we also suggest and deploy the attack countermeasures to harden the blockchain systems security.

Dissertation Contents. The dissertation uses contents from three papers published and three papers in submission by the author. Chapter 3 incorporates material from Reference [93] which presents the static and dynamic analysis of in-browser cryptojacking. Chapter 4 is based on Reference [95] which counters DDoS attacks on blockchain memory pools. Chapter 5 is based on Reference [92] which presents the partitioning attacks on the Bitcoin network. Finally, Chapters 6, 7, and 8 are based on three manuscripts that are currently in submission.

CHAPTER 2: LITERATURE REVIEW

In the following, we discuss the notable related works relevant to this dissertation. In keep with the structure, we first discuss the attacks related to the blockchain application rules (§2.1), followed by the attacks related to the network layer (§2.2).

2.1 Application-specific Attacks

In this section, we discuss the prior works related to cryptojacking and DDoS attacks on blockchain systems. These works are related to §3 and §4 in this dissertation.

In-browser Cryptojacking. The first notable work on in-browser cryptojacking was conducted by Eskandari *et al.* [38] who looked into the prevalence of cryptojacking, showing the use of *Coinhive* as the most popular platform. Concurrently, Rüth *et al.* [90] studied the prevalence of cryptojacking by analyzing blacklisted sites from the No Coin web extension. They mapped those sites on a large corpus of websites obtained from the Alexa Top 1M list, and found 1,491 suspect websites involved in cryptojacking. However, they did not perform static or dynamic analysis of cryptojacking scripts to study the intricacies such covert mining practices. A more systematic treatment of in-browser cryptojacking was performed by by Hong *et al.* [56] and Konoth *et al.* [65]. Hong *et al.* performed static analysis on 2,770 cryptojacking websites and developed a machine learning tool called *CMTracker* that detects and prevents cryptojacking. Concurrently, Konoth *et al.* [65] performed a code-based analysis on 13 cryptojacking platforms to analyze various features in cryptojacking *JavaScript* code and develop countermeasures for it. Later, Kharraz *et al.* [62] presented *Outguard*; a cryptojacking detection tool that uses supervised learning to accurately detect covert mining operations with ≈97% accuracy. However, they did not perform dynamic analysis of cryptojacking scripts to analyze their effect on the user devices.

In the domain of dynamic analysis, Tahir *et al.* [105] presented a tool called *MineGuard*, that performed a real-time detection of covert mining operations in the cloud. *MineGuard* used hardware-

assisted profiling to create discernible signatures for mining algorithms and later use it for detection. Extending their analysis to the in-browser cryptojacking [105], they developed a browser extension that used fine-grained micro-architectural footprint to detect cryptojacking. In our dynamic analysis, we take a different approach to perform a resource profiling and analyze the effect of cryptojacking on user devices. We further look into the semantics of traffic exchange during mining operations and use them to develop effective countermeasures for the real-time detection.

DDoS Attacks. Distributed denial-of-service (DDoS) attacks have been quite prevalent [107]. DDoS attacks are repeatedly launched against the mining pools, the legitimate users, and the currency exchanges. Johnson *et al.* [59] performed a game-theoretic analysis of DDoS attacks against Bitcoin mining pools. Vasek *et al.* [107] empirically illustrated the denial-of-service attacks on the Bitcoin system. Prior to its release on November 12, 2017, Bitcoin Gold suffered from a massive DDoS attack [73]. Cryptocurrency exchanges have also been frequently targeted to prevent coin tradings, and no clear nor specific mitigation techniques to those attacks have been proposed.

Another form of DDoS attack on blockchain includes spamming the network with low valued dust transactions. This attack is also called the *penny-flooding* attack. Baqer *et al.* [7] performed Bitcoin stress testing to analyze this attack and proposed its countermeasures.

2.2 Network Layer Attacks

In this section, we discuss prior works that are related to our work in §5, §6, and §7. Note that some of these attacks are intrinsic to the blockchain systems in general. Our work shows how the network inconsistencies can be exploited to amortize the cost of these attacks.

The 51% Attack. The 51% attack is a classical weakness in blockchains where an adversary acquires a majority of the network's hash rate to gain control over the blockchain [39, 37]. The 51% attack primarily relies on the ability to generate the "longest chain" in the long run [8], using which the adversary can also perform selfish mining and double-spending, discussed below.

Selfish Mining. Selfish mining is a form of block withholding attack, in which the adversary computes a block and does not publish it [88]. Instead, it keeps on extending its private chain in hopes to attain a longer chain than the competing public chain. When the adversary achieves that, it releases its longer private chain. In Bitcoin, nodes switch to the chain with the longer prefix, thereby invalidating the public chain computed by the honest miners. If the adversary has 51% hash rate, its private chain will eventually be longer than the rest of the network, therefore guaranteeing a successful attack. The problem of selfish mining has been addressed by Eyal and Sirer [99], Sapirshtein *et al.* [99], and Solat and Potop-Butucaru [103].

Double-spending. Double-spending or equivocation is when an attacker spends their cryptocurrency token twice [58]. The double-spending attack is launched in various ways. One possible method is that the attacker sends the transaction to a receiver and the receiver delivers a product before the transaction is confirmed. The attacker then sends the other transaction to himself. Both transactions are received by a miner, who can only accept one of them. Therefore, with 0.5 probability the recipient could be tricked. The other strategy could be that the attacker transacts with the recipient and the transaction gets confirmed in the public blockchain. The attacker then generates the other transaction, adds it to the private blockchain, and launches a selfish mining attack. If the selfish mining succeeds (with probability 1 if the attacker has 51% hash rate), then the recipient's transaction will be invalidated along with the public blockchain.

Please note that the assumption about the 51% hash rate is only valid in a *lock-step* synchronous or *non-lock-step* synchronous network in which a block experiences negligible propagation delay [44, 83]. If the network exhibits asynhrony or the block propagation delay becomes significant, then all the attacks asscoiated with the 51% hash rate can be amortized.

Block Withholding Attack. Block Withholding Attack was presented by Rosenfeld [87] in which miners in a pool choose to submit partial proof of work, instead of the full proof. As a result, they get rewarded for participating in the pool although the pool suffers a loss due to partial solutions. Kwon *et al.* [67] studied a new attack on blockchains called "Fork After Withholding" (FAW)

attack that guarantees higher rewards than block withholding attacks.

Since §5, §6, §7 include Bitcoin network measurements, it is therefore important to mention prior notable measurement studies that were conducted along the same lines.

Bitcoin Network Measurements. Notable works on the Bitcoin network measurements have focused on (1) analyzing Bitcoin nodes distribution across autonomous systems (ASes) [5, 45], (2) discovering influential nodes in mining pools [35, 5], and (3) measuring the Bitcoin block propagation[34]. In 2013, Decker *et al.* [34] conducted the first measurement study to analyze the information propagation in the Bitcoin network. They concluded that the block size is the dominant factor in block propagation delay. In their measurements, they connected to ≈3K IP addresses and observed that ≈90% of the nodes in the network receive the newly published block within 12.6 seconds on average. In §5–§7, we will further elaborate on these measurement studies.

CHAPTER 3: STATIC AND DYNAMIC ANALYSIS OF IN-BROWSER CRYPTOJACKING[1]

Notable blockchain-based cryptocurrencies use the Proof-of-Work (PoW) consensus protocol to mine blocks through extensive hash operations. Since mining is a resource-intensive task, therefore, it incurs a significant cost. One way to circumvent the cost is by hijacking machines of other users and use them for mining. This technique is called cryptojacking, where a target device is used to mine the cryptocurrency on behalf of an adversary. A recent form of cryptojacking is called in-browser cryptojacking in which *JavaScript* code is used to compute PoW in a web browser and transmit the PoW to a remote server controlled by the adversary [102]. Since the cryptojacking scripts are executed in the browser, therefore, they are not detected by the antivirus scanners. The existing countermeasures of in-browser cryptojacking include using a blacklisting approach to block websites that use the cryptojacking scripts. However, blacklisting can easily be circumvented by using proxy servers, therefore creating a need for more robust countermeasures.

3.1 Contributions

In this work, we conduct static and dynamic analyses of in-browser cryptojacking to understand its operations and impacts. By using insights from our analyses, we propose robust countermeasures that outperform the existing methods. Our key contributions are summarized below.

1. Using more than 5,700 websites with cryptojacking scripts, we conduct static analysis to identify the distribution of cryptocurrencies used in cryptojacking, and study code complexity features of cryptojacking scripts (§3.3). Using those features, we build an unsupervised clustering system that automatically identifies cryptojacking, malicious, and benign scripts with ≈96% accuracy. (§3.3.3).

[1]This content was reproduced from the following article: M. Saad, A. Khormali, and D. Mohaisen. Dine and Dash: Static, Dynamic, and Economic Analysis of In-browser Cryptojacking. *In APWG Symposium on Electronic Crime Research*, pages 1–12, 2019.

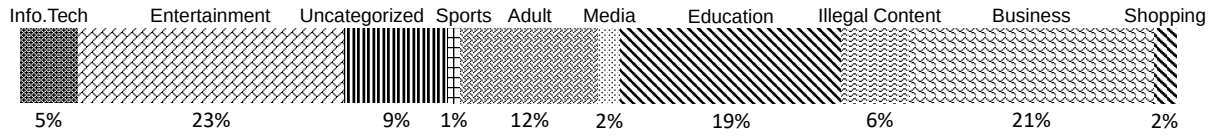


Figure 3.1: Website categorization based on the main topic. Note that most websites belong to Entertainment, Business, and Education. A sizable chunk (12%) belonged to the Adult category.

2. We perform dynamic analysis to observe CPU usage and network usage during cryptojacking (§3.4). Using insights from the network usage (WebSocket communication), we propose simple and effective techniques to counter cryptojacking in the web browser (§3.5).

3.2 Preliminaries and Data Collections

In-browser cryptojacking is done by injecting a *JavaScript* code in a website, allowing it to hijack the processing power of a visitor's device. Generally, *JavaScript* is automatically executed when a website is loaded. Upon visiting a website with cryptojacking code, the visiting host starts a mining activity, by becoming part of a cryptojacking mining pool. A key feature of in-browser cryptojacking is being platform-independent: it can be executed on any host, PC, mobile phone, tablet, etc., as long as the web browser running on this host has *JavaScript* enabled in it.

For our static analysis, we assembled a data set of cryptojacking websites published by Pixalate [70] and Netlab 360 [112]. Pixalate is a network analytics company that provides data solutions for digital advertising and research. In Nov. 2017, they published a list of 5000 cryptojacking websites that were actively stealing their visitor's processing power to mine cryptocurrency. Netlab 360 (Network Security Research Lab at 360) is a data research platform that provides a wide range of datasets spanning Domain Name Servers (DNS) and Distributed Denial-of-Service (DDoS) attacks. From Netlab 360, we obtained 700 cryptojacking websites.

3.3 Static Analysis

In static analysis, we pursue three directions: content-, currency-, and code-based analysis. Content-based categorization provides insights into the nature of websites used for cryptojacking, while the currency-based categorization shows platforms used for it. The code-based analysis provides insight into the complexity of the cryptojacking scripts, using code complexity measures.

3.3.1 Content and Currency-based Categorization

We categorized the websites based on their contents into various categories using the *WebShrinker* website URL categorization API. *WebShrinker* assigns categories to websites based on the content present in those websites. The results are presented in Fig. 3.1, showing that miners have utilized a wide range of categories for in-browser cryptocurrency mining, including education, business, entertainment, etc.. Notice that 19% websites were categorized as "Education" which can be attributed to the exploitation of trust by adversaries behind cryptojacking [113].

By analyzing our dataset, we found eight platforms providing templates to mine two types of cryptocurrencies: Monero and JSEcoin. Table 3.1 provides details of those platforms. We found that a large percentage of the websites ($\approx$81.57%) used *Coinhive* [13] to mine Monero cryptocurrency [28]. Additionally, $\approx$2.61% websites used the JSEcoin platform [27], which is mines the JSEcoin cryptocurrency. Therefore, we detected two cryptocurrencies used in cryptojacking.

3.3.2 Code-based Analysis

For code-based analysis, we gathered cryptojacking scripts from all the major cryptojacking service providers found in our dataset, such as *Coinhive*, JSEcoin, Crypto-Loot, Hashing, deepMiner, Freecontent, Miner, and Authedmine. We observed that all the service providers had unique codes, specific to their own platform. In other words, the websites using *Coinhive*'s services had the same *JavaScript* code template across all of them. Therefore, $\approx$81.57% of the websites in our dataset

Table 3.1: Currency-based analysis results. [1] The abbreviation No CJ means No cryptojacking.

Platform	Websites		Cryptocurrency	Websites	
	#	%		#	%
Coinhive	4652	81.57	Monero	4926	86.37
Hashing	67	1.17			
deepMiner	56	0.98			
Freecontent	39	0.68			
Cryptoloot	38	0.67			
Miner	38	0.67			
Authedmine	35	0.61			
JSEcoin	149	2.61	JSEcoin	149	2.61
No CJ	628	11.01	—	628	11.01
Total	5703	100.00	—	5703	100.00

were using the same *JavaScript* template for cryptojacking. Similarly, all the websites using JSEcoin used the same standard template for their mining. However, the code template of each service provider was different from one another, which led us to believe that each script had unique static features. With this in mind, we performed the code-based analysis on the cryptojacking websites and compared the results with other standard *JavaScript* for a baseline comparison.

Data Attributes. We prepared our dataset for static analysis by collecting all of the popular cryptojacking scripts from our list of websites. As a control experiment, we collected an equal number of malicious and benign *JavaScript* codes to design a clustering algorithm. Our aim was to obtain a set of features that were unique only to the cryptojacking scripts, and aid in their detection.

To avoid bias towards a certain class, we were limited to include equal size of malicious and benign *JavaScript* samples for the static analysis. Although there are many samples of malicious and benign *JavaScript* in the wild [32], only eight cryptojacking scripts are available in comparison. Since our work is focused on distinguishing cryptojacking scripts from both malicious and benign *JavaScript*, we had to balance the size of each class. While the number of scripts might seem as a limitation of our work, we believe the promise of this work is substantial: as more currencies and platforms start to use cryptojacking, more samples will be available for a broader study.

In lieu, we used the existing data of the cryptojacking websites (§3.2) and online resources from

GitHub for malicious *JavaScript* sample [111] . For benign *JavaScript*, we used the set of non-cryptojacking websites and parsed their HTML code to extract benign *JavaScript* code [104]. In summary, we had 8c̃j *JavaScript* samples, spanning all the websites. Accordingly, we selected 10 malicious and 10 benign scripts for our clustering analysis.

Feature Extraction. We use various features that provide insights into the code structure and its maintainability. The features that we extracted include (1) cyclomatic complexity M [110], (2) Control Flow Gaph (CFG) features including number of nodes N, edges E, and connected components Q, (3) cyclomatic complexity density M_d [43], (4) total number of lines of code c_l, (5) Halstead complexity measures including vocabulary η, program length n, calculated program length n_c, volume V, effort E, delivered bugs B, time T, difficulty D, distinct operators η_1, distinct operands η_2, number of operator n_1, number of operands n_2, Halstead volume V, source lines of code $sloc$, and maintainability score M_s. We extracted these features using *Plato*, a *JavaScript* static analysis tool [6]. We report results in Table 3.2 where it can be observed that certain features, such as M, M_d, V, and T, are clearly discriminative among all the categories.

3.3.3 Fuzzy C-Means Clustering

In this section, we build a classification system that automatically recognizes cryptojacking scripts from malicious and benign scripts based on the code complexity features alone, which could be easily extracted from the cryptojacking scripts and are common among a large number of cryptojacking websites. It is desirable for our classification system to classify scripts even with minimal information regarding the labels of the scripts. Therefore, we utilized the Fuzzy C-Means (FCM) clustering algorithm [9], which has the advantage of being an unsupervised learning algorithm. In the other words, in comparison with supervised classification algorithms, such as the Support Vector Machine (SVM) and Random Forest (RF), which require labels of the dataset in the training phase, FCM has the advantage of performing well on the unlabeled dataset.

We utilized the FCM clustering algorithm to group the scripts to cryptojacking, malicious, and benign clusters. In order to evaluate the performance of the clustering experiment, we used stan-

Table 3.2: Features values for cryptojacking, malicious, and benign samples.

Cat.	Platforms	M	M_d	B	D	E	c_l	T	η	V	η_1	n_1	η_2	n_2	params	sloc	physical	M_s
Cryptojacking	deepMiner	184	44.2	14.1	113.0	4,810,434	4,667	267,246	554	42,533	47	2,440	507	2,227	75	416	499	67.8
	Authedmine	168	26.5	19.7	82.8	4,912,255	6,096	272,903	844	59,259	41	3,247	803	2,849	73	633	784	62.8
	Hashing	138	29.1	7.2	94.6	2,185,379	2,794	124,138	342	24,393	38	1,469	315	1,415	37	412	505	68.2
	Miner	133	27.7	9.3	90.5	2,537,930	3,239	140,996	403	28,032	39	1,690	364	1,549	49	479	617	64.1
	Coinhive	131	27.5	9.1	94.8	2,608,021	3,226	144,890	368	274,970	37	1,697	331	1,529	48	476	594	63.7
	Crypto-loot	128	39.7	11.4	88.1	3,034,935	3,788	168,607	546	34,443	45	1,962	501	1,826	62	322	389	70.3
	Freecontent	117	28.3	8.1	89.4	2,180,394	2,884	121,133	350	24,373	38	1,469	312	1,415	37	412	505	62.7
	JSEcoin	64	17.2	10.2	62.9	1,945,165	3,257	108,064	716	30,888	45	1,878	671	1,379	49	372	412	64.7
	Mean (μ)	130.3	29.9	11.3	88.9	3,026,191	3,755.1	168,121	516.4	33,925	41.3	1,981.5	475.1	1,773.6	53.8	440.3	538.1	64.9
	SD. (σ)	35.9	8.4	3.9	13.8	1,180,403	1,109.9	65,577	185.1	11,856	3.9	599.3	182.8	519.3	14.8	93.2	126.3	2.8
Malicious	20160209	92	21.5	5.6	25.1	423,925	1,833	23,551	580	16,826	27	1,032	553	801	22	427	503	44.4
	20161126	62	15.3	4.2	24.6	315,735	1,563	17,540	292	12,800	17	798	275	765	0	403	481	90.5
	20170110	14	4.4	15.0	26.7	1,211,305	4,704	67,294	782	45,210	15	2,740	767	1,964	232	313	564	93.6
	20170507	6	24.0	5.9	11.1	199,917	1,864	11,106	777	17,897	18	942	759	922	1	25	890	71.7
	20160927	3	1.4	4.0	32.5	393,555	1,575	21,864	204	12,084	13	957	191	618	0	213	98	23.2
	20170322	2	18.1	11.8	7.1	253,442	3,514	14,080	1,123	35,607	9	1,762	1,114	1,752	3	11	1,738	90.9
	20170303	2	8.6	0.2	9.4	8,338	147	463	63	878	13	73	50	74	4	23	55	78.7
	20160407	1	33.3	0.1	2.7	207	19	11	16	76	5	12	11	7	0	3	3	78.9
	20170501	1	0.9	2.1	3.3	21,464	758	1,192	322	6,314	5	431	317	327	0	105	105	35.9
	20160810	1	12.5	0.5	11.9	20,148	275	1,119	70	1,685	6	255	64	20	0	8	13	60.4
	Mean (μ)	18.4	14	4.9	15.5	284,803.7	1,625.2	15,822	422.9	14,938	12.8	900.2	410.1	725	26.2	153.1	445	66.9
	SD. (σ)	31.9	10.5	5	10.8	364,470.8	1,508.9	20,248	374.8	15,045	6.9	834.7	372.5	686.6	72.6	171.9	543.5	24.9
Benign	The Boat	2,135	69.3	110.8	392.0	130,285,522	31,916	7,238,084	1,364	332,361	59	17,341	1,305	14,575	852	3,084	3,349	66.7
	IBM Design	2,119	68.3	110.9	397.1	132,237,213	32,018	7,346,511	1,351	332,981	59	17,393	1,292	1,4625	853	3,103	3,372	66.7
	Histography	1,743	40.7	95.2	249.5	71,325,242	26,627	3,962,513	1,704	285,833	55	14,963	1,649	11,663	803	4,278	5,043	59.4
	Know Lupus	1,006	28.1	92.9	170.4	47,474,425	25,120	2,637,468	2,181	278,600	54	13,424	2,127	11,696	615	3,583	4,288	65.2
	tota11y	815	38.8	59.4	227.7	40,563,065	17,486	2,253,503	1,167	178,157	52	9,764	1,115	7,722	412	2,099	2,336	62.9
	Masi Tupungato	784	58.2	47.1	185.0	26,199,193	14,296	1,455,510	958	141,585	43	7,875	915	6,421	238	1,347	1,470	67.2
	Fillipo	703	42.9	43.1	194.3	25,139,766	12,900	1,396,653	1,045	129,377	54	7,132	991	5,768	269	1,637	1,770	61.5
	Leg Work	412	75.7	34.0	241.3	24,651,056	11,100	1,369,503	589	102,143	45	5,835	544	5,265	66	544	633	65.9
	Code Conf	409	27.8	41.1	197.1	24,336,420	12,500	1,352,023	939	123,437	49	7,162	890	5,338	315	1,469	1,753	64.9
	Louis Browns	368	35.6	21.2	106.7	6,792,400	6,529	377,355	862	63,667	51	3,393	811	3,136	68	1,034	1,357	53.3
	Mean (μ)	1,049.4	48.5	65.6	236.1	52,900,430	19,049.2	2,938,912	1,216	196,814	52.1	10,428.2	1,163.9	8,621	449.1	2,217.8	2,537.1	63.4
	SD. (σ)	694	17.8	33.6	92.8	44,755,377	9,151.2	2,486,409	459.8	100,856	5.3	4,999	456.7	4,165	310.3	1,225.4	1,418.2	4.3

Table 3.3: Confusion matrix and evaluation metrics of the cryptojacking (CJ), malicious, and benign scripts' clustering results based on FCM clustering algorithm. $^{(1)}$ Evaluation metrics' names are abbreviated. FPR= False Positive Rate, FNR= False Negative Rate, and AR=Accuracy Rate.

Class	Benign	Malicious	CJ	FPR%$^{(1)}$	FNR%$^{(1)}$	AR%$^{(1)}$
Benign	9	0	1	10	0	90
Malicious	0	10	0	0	0	100
CJ	0	0	8	0	11.1	100
Total	9	10	9	3.3	3.7	96.42

dard evaluation metrics; the confusion matrix, Accuracy Rate (AR), False Positive Rate (FPR), and False Negative Rate (FNR), reported in Table 3.3. As shown in Table 3.3, the clustering algorithm is able to classify the scripts with high performance: AR of ≈96.4%, FPR of 3.3%, and FNR of 3.7%. In addition, we have visualized these clusters based on two major principal components of their features, which in Fig. 3.2, clearly show a natural separation between the clusters.

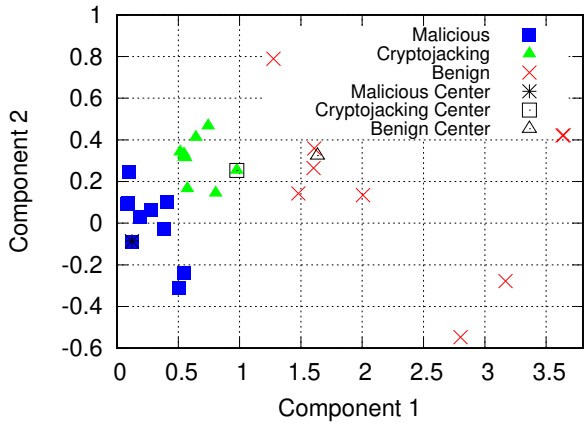


Figure 3.2: Clustering of the cryptojacking, malicious, and benign scripts using FCM.

3.4 Dynamic Analysis

Despite the clear benefits of the static analysis outlined above, it is limited, and subject to circumvention through *JavaScript* code obfuscation. To this end, we conduct dynamic analysis that looks into the impact of cryptojacking on CPU and Network usage.

Settings and Measurements Environment. We noticed that in each cryptojacking website, a *JavaScript* snippet encodes a key belonging to the code owner and a link to a server to which the PoW is ultimately sent. Fig. 3.3 provides a script found in websites that use *Coinhive* for mining. The source (*src*) refers to the actual *JavaScript* file that is executed after a browser loads the script tag. In this script, we also noticed a *throttling parameter*, which is used as a mean of controlling how much resources a cryptojacking script uses on the host. We use such a throttling parameter, α as an additional variable in our experiment. We experiment with $\alpha = \{0.1, 0.5, 0.9\}$.

To understand the impact of cryptojacking on resources usage in different platforms, we use machines running Microsoft Windows, Linux, and Android operating systems (OSes). For our experiments, we selected three laptops, each with one of those OSes. The Windows laptop was Asus V502U, with Intel Core i7-6500U processor operating at 3.16 GHz. The Linux laptop was Lenovo G50, with Intel Core i5-5200U processor (4 cores) running at 2.20 GHz, and the Android phone

```
<script src="./Welcome_files/coinhive.min.js"></script>
<script>
  var miner = new coinhive.Anonymous("owner key",
      {throttle: 0.1});
    miner.start();
</script>
```

Figure 3.3: Malicious JavaScript code that links to Coinhive.

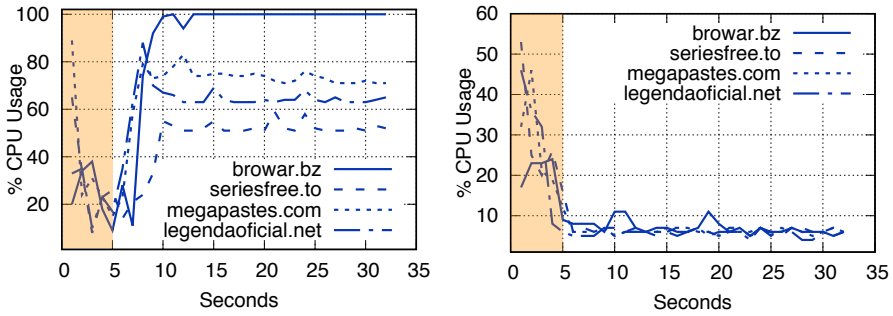
was *Samsung Galaxy J5*, with android version of 6.0.1.

For our cryptojacking script construction, using the various parameters learned above, we set up an account on *Coinhive* to obtain a key that links our "experiment website" to the server. Next, we set up a test website and embedded the code in Fig. 3.3 within the HTML tags of the website. Finally, to measure the usage of resources while running cryptojacking websites, we set up a Selenium-based web browser automation and run cryptojacking websites, for various evaluations. Selenium is a portable web-testing software that mimics actual web browsers [11, 29].

3.4.1 CPU Usage

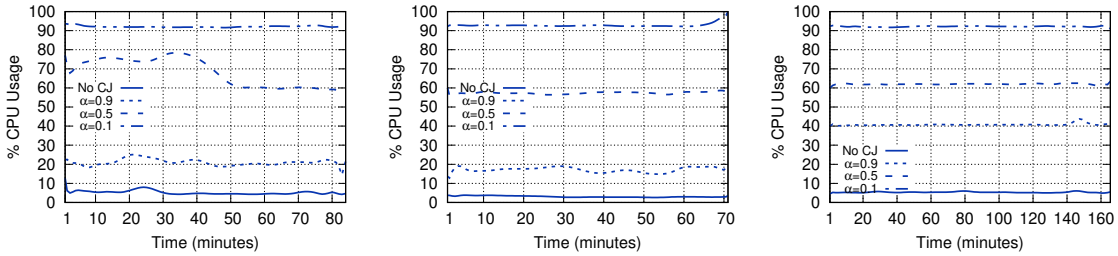
First, we baseline our study to highlight CPU usage as a fingerprint across multiple websites that employ cryptojacking using the aforementioned configurations and measurement environment. We study the CPU usage with and without cryptojacking in place. For this experiment, we select four cryptojacking websites. To measure the impact of cryptojacking on CPU usage, we ran those websites in our *Selenium* environment, for 30 seconds, with *JavaScript* enabled and disabled (baseline; not running the cryptojacking scripts). We use this experiment as our control.

Results. We obtained two sets of results for each website, with and without cryptojacking. In Fig. 3.4, we plot four test samples obtained from our experiment to demonstrate the behavior of websites with and without cryptojacking. From those results, we observe that when a website is loaded initially it consumes a significant CPU power (shaded region), in both cases. Once the website is loaded, the CPU consumption decays if the *JavaScript* is disabled, indicating no cryptojacking.



(a) JavaScript enabled (b) JavaScript disabled

Figure 3.4: Processor usage by four cryptojacking websites with JavaScript enabled and disabled.



(a) Windows CPU usage (b) Linux CPU Usage (c) Android CPU usage

Figure 3.5: CPU usage on devices. As the throttling parameter decreases, the CPU usage increases.

When *JavaScript* is enabled, the CPU consumption is high, indicating cryptojacking. It can also be observed in Fig. 3.4, that the CPU usage varied across the websites, indicating the usage of the throttling parameter highlighted above. The same behavior as with *JavaScript* disabled is exhibited when loading a page with *JavaScript* that is either benign or of other types of maliciousness than cryptojacking. We found that cryptojacking consumes anywhere between 10 and 20 times compared to when not using cryptojacking on the same host.

To understand the impact of throttling on CPU usage in different platforms, we conduct another experiment where we used $\alpha = \{0.1, 0.5, 0.9\}$ with the different testing machines. We found a consistent pattern, whereby the relationship between α and the CPU usage is linear(Fig. 3.5).

3.4.2 Network Usage and Profiling

Dynamic network-based artifacts are essential in analyzing cryptojacking scripts, especially when those scripts are obfuscated. To this end, we also explore the network-level artifacts to reconstruct the operation of cryptojacking services.

We observed that during cryptojacking website execution, the *JavaScript* code establishes a WebSocket connection with a remote server and preforms a bidirectional data transfer. When a WebSocket request is initiated, the client sends an *auth* message to the server along with the user information, including *sitekey*, *type*, and *user*. The length of *auth* message is 112 bytes. The *sitekey* parameter is used by the server to identify the actual user who owns the key of the *JavaScript* and adds balance of hashes to the user's account. The server then authenticates the request parameters and responds back with *authed* message. The *authed* message length is 50 Bytes and it includes a token and the total number of hashes received so far from the client's machine. In the *authed* message, the total number of hashes is 0, since the client has not sent any hashes yet. Then, the server sends *job* message to the client. The *job* message has a length of 234 Bytes with a *job_id*, *blob*, and *target*. The *target* is a function of the current difficulty in the cryptocurrency to be mined. The client then computes hashes on the *nonce* and sends a *submit* message back to the server, with *job_id*, *nonce*, and the resulting hash. The *submit* message has a payload length of 156 Bytes. In response to the *submit* message, the server sends *hash_accept* message with an acknowledgement and the total number of hashes received during the session. The *hash_accept* message is 48 Bytes long. Table 3.4, provides details about the WebSocket connection during cryptojacking.

3.5 Countermeasures

Existing Countermeasures. At the browser level, existing countermeasures include web extensions such as No Coin, Anti Miner, and No Mining [61]. Each of these web extensions maintains a list of uniform resource locators (URLs) to block while surfing websites. If a user visits a website

Table 3.4: Messages exchanged between the client and the server in a WebSocket connection.

Message	**Source**	**Sink**	**Length (Bytes)**	**Parameters**
auth	client	server	112	sitekey, type, user
authed	server	client	50	token, hashes
job	server	client	234	job_id, blob, target
submit	client	server	156	job_id, result
hash_accept	server	client	48	hashes

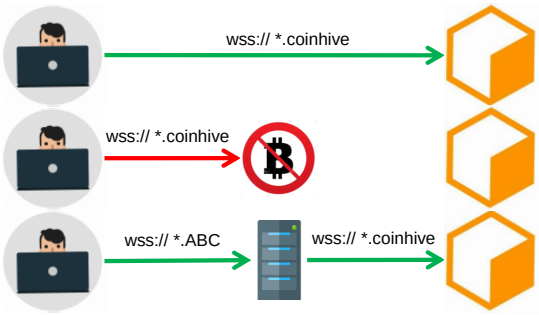


Figure 3.6: Circumventing detection by relaying WebSocket requests through a proxy server.

that is blacklisted by the extension, the user is notified about cryptojacking. However, we show that blacklisting is not effective to counter cryptojacking since an adaptive attacker can circumvent detection by creating new links that are not found in the public list of blacklisted URLs.

Evading Detection. An attacker can evade detection by setting his own third party server to relay data to and from cryptojacking server. In Fig. 3.6, we show how the current countermeasures for cryptojacking can be circumvented. To practically demonstrate that, we set up a test website using *Coinhive* script and installed a local relay server. We installed four chrome extensions that block the in-browser cryptojacking: No Coin, Anti Miner, No Mining, and Mining Blocker. In the first phase, we installed the *Coinhive* script and ran the website. Each extension detected the WebSocket request and blocked it. We then configured our relay server to act as a proxy. In the *Coinhive* script, we modified the code and replaced the *Coinhive* socket address with our server address. Next, when we visited the website, it started cryptojacking on the client machine.

Robust Countermeasures. Instead of blocking specific URLs, the extensions can monitor the messages exchanged between the user and the server during cryptojacking session. If the messages follow the sequence of web frames illustrated in Table 3.4, the extension can flag them as cryptojacking. This will prevent cryptojacking even if WebSocket requests are relayed through a proxy. We developed a chrome web extension that detects the strings of web frames shown in Table 3.4, and notifies the user when the website starts cryptojacking. To test our extension against the existing countermeasures, we deployed a proxy server that relayed the data between our test website to the *dropzone* server as shown in Fig. 3.6. Our web extension immediately flagged cryptojacking upon reading the data exchanged between the browser and the relay server.

3.6 Summary

This work demonstrates how the PoW implementation can be exploited to abuse resources of online users through in-browser cryptojacking. Towards that, we systematically analyzes in-browser cryptojacking through the lenses of static and dynamic analyses. Our static analysis unveils unique code complexity characteristics and can be used to detect cryptojacking code from malicious and benign code samples with $\approx$96% accuracy. Our dynamic analysis shows the CPU usage in cryptojacking and we use that knowledge to reconstruct the operation of cryptojacking scripts. Finally, by surveying prior countermeasures and examining their limitations, we show simple and effective methods to counter cryptojacking, capitalizing on the insights from our dynamic analysis.

This is to be noted that in-browser cryptojacking is enabled by resource intensive PoW protocol which is an application design choice made by the cryptocurrency creators. If Monero or JSEcoin were using the Proof-of-Stake (PoS) protocol, then the in-browser cryptojacking would not have been resource-intensive. Therefore, the design choice for a cryptocurrency application has security implications for other users, which we have thoroughly investigated in this chapter.

CHAPTER 4: COUNTERING DDOS ATTACKS ON BLOCKCHAIN MEMORY POOLS[1]

In this chapter, we identify and counter a new DDoS attack on the blockchain memory pools that increase the transaction mining fee. Blockchain-based cryptocurrencies set application-specific policies for transaction processing [1]. In Bitcoin, for example, the block size is limited to 1MB and the average block time is set to 10 minutes. As a result, Bitcoin can only process up to 7 transactions per second. Moreover, in blockchain-based cryptocurrencies, a memory pool (mempool) is a repository for unconfirmed transactions where transactions stay before being mined in a block. At the mempool, if the rate of incoming transactions exceeds the network throughput, a transaction backlog develops which causes users to pay a higher mining fee to prioritize their transactions. We note that this behavior can be exploited to launch a denial-of-service attack against users by flooding the mempool with dust transactions and forcing them to pay a higher mining fee. Current blockchain-based cryptocurrencies do not apply any mechanism to prevent such an attack.

4.1 Contributions

In this chapter, we continue our analysis on attacks related to application-specific policies. Towards that, we identify a mempool DDoS attack that forces benign users to pay higher mining fee. We propose two attack countermeasures and evaluate their performance using discrete-event simulations. Our key contributions are summarized below.

1. We identify the effect of mempool flooding on benign users, leading up to a DoS attack.

2. We present a threat model and associated attack procedure whereby an attacker can exploit the current Bitcoin protocol to achieve his goals.

[1]This content was reproduced from the following article: M. Saad, L. Njilla, C. A. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen. Mempool Optimization for Defending Against DDoS Attacks in PoW-based Blockchain Systems. In *IEEE International Conference on Blockchain and Cryptocurrency*, pages 285–292, 2019

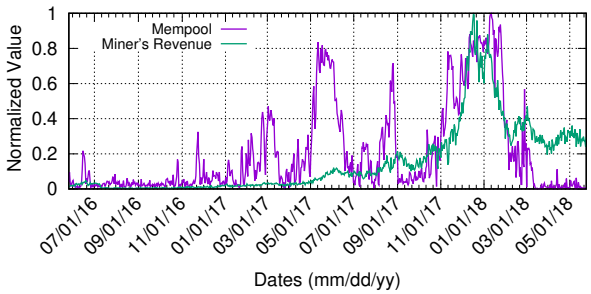


Figure 4.1: Relationship between the mempool size and the mining fee paid by the users. Notice that as the mempool size grows, the mining fee increases accordingly. The spikes during May, September, and November 2017 indicate spam attacks.

3. To counter the attack, we propose fee-based and age-based countermeasures. We examine the performance of our proposed countermeasures through discrete-event simulations and evaluate their performance under varying attack conditions.

4.2 Background and Preliminaries

UTXO. In Bitcoin, a user generates a transaction by using spendable balance in his wallet. Spendable balance comprises of confirmed "Unspent Transaction Outputs" (UTXO's) [94, 97] that are previously mined in the blockchain.

Relay Fee and Mining Fee. In Bitcoin, relay fee is the minimum fee paid for a transaction to be included in a mempool. If a transaction does not pay the relay fee, peers do not relay it to other nodes. Mining fee is the fee paid to a miner as an incentive to include the transaction into a block [20]. Miners tend to prioritize the transactions that pay higher mining fee.

Confirmation. Transaction confirmation means that a transaction has been mined into a block and its parent UTXO's are valid and spendable in receiver's wallet [20]. A confirmation score, also known as the age of a transaction, is the difference between the block number in which it was mined and the most recent block. A confirmation score of 0 means that the transaction is in the mempool, and not yet mined. Such a transactions is also called an "unconfirmed transaction."

Memory Pool. In cryptocurrencies, a memory pool (mempool) is a cache of unconfirmed transactions [20]. The mempool is a bottleneck in the system, and if the transaction arrival rate exceeds the mining rate, the mempool size starts to grow and the verification process gets delayed.

Dust Transactions. In cryptocurrencies, transactions with small input values are known as "dust transactions" [66]. Dust transactions contribute very little to the exchange volume of Bitcoin but consume as much space in the block as a high valued transaction. Spam attacks to exhaust the block space are carried out using these transactions [7].

DDoS Attack on Mempools. The DDoS attack presented in this work targets the blockchain mempools by flooding them with unconfirmed transactions. Although, these transactions may eventually be rejected by miners, but their presence in the mempool creates another major problem. The mempool size determines the fee paid to the miners. If the mempool size is big, miners have a limited choice of mining the transactions, and the users try to prioritize their transactions by paying higher mining fee. Therefore, by mempool flooding, the attacker might trap the users into paying high fee. Fig. 4.1, shows the relationship between the mempool size and the fee paid to the miners.

Data Collection. To observe the relationship between the mempool size and the mining fee, we used the public dataset provided by the company called "Blockchain" [17]. In Fig. 4.1, we plot the results obtained from the dataset and we use the min-max normalization to scale our dataset in the range [0–1]. Our data shows that Bitcoin mempool has been attacked three times in 2017, and each time it resulted in an increased mining fee, supporting the attack premise.

4.3 Threat Model

For our threat model, we assume an attacker with spendable balance in this wallet. The attacker controls a group of sybil accounts, each with multiple public addresses, intended to be used during the attack. Furthermore, the attacker and sybils have client side software and scripts, which enable them to initiate a flood of "raw transactions" [20], higher rate than the network throughput [31].

Additionally, the attacker is also constrained by a fixed "budget." Since each transaction requires a minimum relay fee, it limits the number of transactions that the attacker can generate.

Attack Objectives. The attacker's objective is to flood mempools with unconfirmed *dust transactions*. At mempools, the arrival rate corresponds to the flow of incoming transactions and the departure rate corresponds to the rate of transaction mining. The departure rate is fixed, because the average block computation time and the size of the block are fixed. When the arrival rate increases due to a flood of transactions, it results in transactions backlog. Overwhelming the mempool size alarms the legitimate users, who naturally start paying higher mining fee to prioritize their transactions. The secondary objective of the attacker will be to reduce the attack cost by getting his transactions rejected. For the attacker, mining will result into losing balance to miners. However, if the transactions get rejected, the attacker will have another chance to repeat the attack.

4.3.1 Attack Procedure

To reduce the attack cost, the attacker will design his transactions in a way that they are less likely to be prioritized by miners. At the same time, the attacker would want his transactions to stay in mempools for as long as possible. To this end, we envision that this attack can be carried out in two phases: the distribution phase and the attack phase.

The Distribution Phase. In the distribution phase, the attacker estimates the minimum relay fee of the network, divides his spendable bitcoins ("UTXO's") into various transactions and sends them to the sybil accounts. The attacker generates a series of outputs to all the addresses of sybil nodes with one or more transactions per address. Transactions made in the distribution phase will have input "UTXO's", which will be previously mined in the blockchain. Hence, these transactions will have greater-than-zero age, and will be capable of paying the relay fee and the mining fee.

The Attack Phase. In the attack phase, sybils will carry out "raw transactions" [20] from the balance received in the distribution phase. Sybils will generate dust transactions and exchange them with each other. The rate of exchange of transactions will be much higher than the network throughput. As a result, the arrival rate of the transactions at mempools will be higher than the

departure rate of mined transactions. This will increase the transaction backlog and the mempool size. Transactions made in the attack phase will have transactions of distribution phase as input "UTXO's". These inputs will still be awaiting confirmation in the blockchain. Due to that, their confirmation factor or age score will be zero.

4.3.2 Attack Cost

To reduce the attack cost, the attacker requires transactions to be part of the mempool but not the blockchain. This can be achieved by adding the minimum relay fee (R_f) to each transaction but not the minimum mining fee (M_f). The relay fee is necessary for a transaction to propagate in the network and be accepted by the mempool. If the attacker adds the mining fee, his transactions will attain priority from a miner and might get mined. To avoid that, sybils only pay the relay fee. If a transaction has i inputs, each contributing a size of k Bytes, and o outputs, each contributing a size of l Bytes, then the total transaction size S and its cost C(BTC) are determined by (4.1).

$$S(\text{Bytes}) = (i \times k) + (o \times l) + i, C = R_f \times \frac{S}{1024} \tag{4.1}$$

Assuming that the attacker is limited by a budget B (BTC) and minimum transferable value set by the network as $T_{\min}$, then the total number of transactions T_a that the attacker can generate can be computed in (4.2).

$$T_a = \frac{B \times 1024}{R_f \times T_{\min} \times [(i \times k) + (o \times l) + i]} \tag{4.2}$$

On the other hand, a legitimate user who intends to get his transaction mined, pays relay fee for transaction broadcast and mining fee as an incentive to the miner. For such a user, contributing a total T transactions, the cost incurred per transactions and the total cost of all transactions T_l is:

$$C = [R_f + M_f] \times \frac{S}{1024}, T_l = \boldsymbol{T} \times [R_f + M_f] \times \frac{S}{1024} \tag{4.3}$$

Under these settings, the maximum loss an attacker can incur would happen if all his transactions possibly get mined. The cost in such a case will be the product of the total number of transactions and the relay fee ($T_a \times R_f$). The attacker can re-launch the same attack with a new balance of $B - (T_a \times R_f)$. If a portion of the attacker's total transactions t_a gets mined, where $t_a \leq T_a$, then the attacker would be able to re-launch the attack with new balance of $B - (t_a \times R_f)$.

4.4 Countering The Mempool Attack

To counter DDoS on Bitcoin's mempool, we propose fee-based and age-based countermeasures. In the following, we provide a detailed description of both designs along with experimental results.

4.4.1 Fee-based Mempool Design

For fee-based mempool design, we assume that the mempool is initially empty when transactions begin to arrive. We fix a baseline threshold beyond which the mempool starts spam filtering. Initially, when the transactions arrive at the mempool, for each transaction, the mempool checks if the transaction pays a minimum relay fee. If the transaction pays the minimum relay fee, it is accepted and the mempool size is updated. When the mempool size reaches the threshold, it starts applying the fee-based policy. Now, if the incoming transaction pays both the relay fee and the mining fee, only then it is accepted. As a result, we filter spam transactions to optimize the mempool size. If the new size is less than the baseline threshold then the mempool proceeds its operation from relay fee check. Otherwise, it continues with the fee-based design.

Analysis of Fee-based Mempool Design. In the following, we analyze the workings of fee-based design and its utility in the light of our threat model. We will limit the number of transactions an attacker can generate within his budget by increasing the mining fee threshold. We also observe how this design affects legitimate users in the network.

In the current settings, if mempools employ the fee-based design, the attacker will add mining fee to each transaction. Given a budget B, adding mining fee to each transaction reduces the total number of attacker's transactions T_a, and the (4.2) will change to:

$$T_a = \frac{1024 \times B}{[(i \times k) + (o \times l) + i] \times [R_f + M_f] \times T_{\min}} \tag{4.4}$$

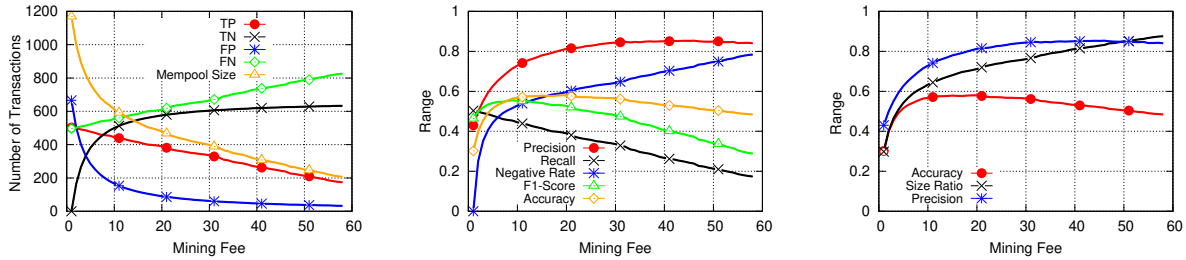
From (4.4), we can observe that the number of transactions the attacker can generate has an inverse relationship with the total fee paid per transactions. Using that relationship, we can adjust the fee parameter and investigate how it limits the attacker's capabilities. To do that, we simulate the affect of increasing the mining fee on the volume of transactions that the mempool accepts. We allocate a fixed budget to the attacker and select thresholds of minimum mining fee and maximum mining fee. Using (4.2), we select a suitable budget for attacker that results into 1,000 transactions with a minimum mining fee. Then, we generate 1,000 legitimate transactions, each with a mining fee normally distributed over the range of the minimum and maximum mining fee. Using discrete-event simulations, we increase the mining fee and monitor its affect on the transactions generated by the attacker and the legitimate users in the network.

Evaluation Results. We plot the results in Fig. 4.2, and use the confusion matrix in Table 4.1, and evaluation parameters Table 4.1 to observe the effect of the fee-based design on the mempool. The results in Fig. 4.2(a) show that with the increase in the mining fee threshold, the mempool size (TP+FP), malicious transactions (FP) and legitimate transactions (TP) decrease. The trend of (FP) is explained by (4.4). With a fixed budget, increasing the mining fee decreases the total transactions T_a. Accordingly, the size of the mempool also decreases due to fewer spam transactions (FP). However, increasing mining fee also limits the budget of legitimate users, which explains the trend of decreasing (TP). Fig. 4.2(b), shows that the accuracy increases with the mining fee to a maximum value and then decreases. Using that, we found a minimum fee cutoff corresponding to the maximum accuracy. In Fig. 4.2(c), we plot accuracy and size ratio; the size ratio is the fraction of mempool transactions out of the total number of incoming transactions. Lower size ratio indicates higher size optimization. The results show that at a fee threshold of 13, we achieve

Table 4.1: Confusion Matrix and Evaluation Parameters

		Actual Transaction	
		Legitimate	Malicious
Mempool Transaction	Legitimate	TP	FP
	Malicious	FN	TN

Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F_1	$\frac{2\times precsion\times recall}{precision+recall}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Negative Rate	$\frac{TN}{TN+FN}$

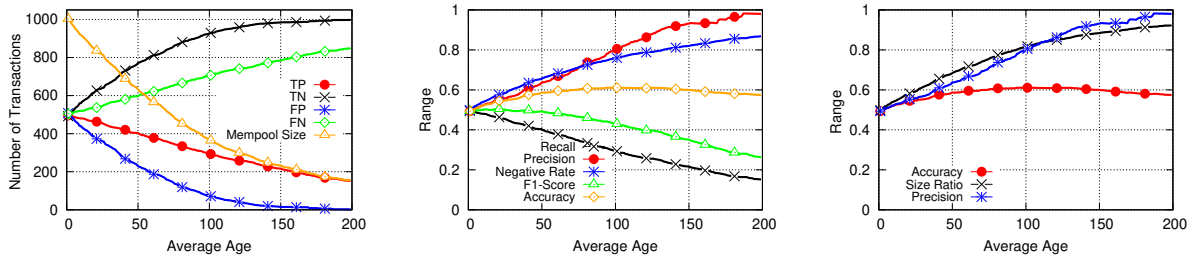


(a) Confusion Matrix (b) Evaluation Parameters (c) Accuracy, Precision, and Size

Figure 4.2: Fee-based design analysis. As the mining fee increases, the mempool size reduces. However this also affects legitimate transactions thereby reducing the detection accuracy. An optimum fee cut-off can be selected from Fig. 4.2(c) based on the accuracy and size ratio trade-off.

60% accuracy, 70% size optimization, and 78% precision. Increasing the fee parameter further, increases the size optimization but decreases the accuracy. Therefore, the fee-based design presents a trade-off between the size efficiency and the accurate detection of malicious transactions.

Limitations of Fee-based Mempool Design. The attacker can circumvent the fee-based design by using a transaction generation technique called "Child Pays For Parent" (CPFP) [20]. For transactions generated in the attack phase, their parent transactions in the distribution phase need to be verified and mined. The attacker can minimize the probability of transaction acceptance in the first phase by reducing their priority factor; *e.g.* by paying a minimum relay fee and no mining fee. Once the parent transactions have lower probability of acceptance in the first phase, the child transactions can increase their priority factor by adding higher relay fee and mining fee. In such a situation, and when the mempools apply the countermeasures, spam transactions of the attack phase will get into the mempool. One way to address this problem is to prioritize the incoming transactions on the basis of the mining fee. However, this will also affect the legitimate transactions. To address these limitations we propose the age-based countermeasures.



(a) Confusion Matrix (b) Evaluation Parameters (c) Accuracy, Precision, and Size

Figure 4.3: Analysis of the age-based Design. Notice that with age-based design, the accuracy, precision, and size ratio are comparatively higher than the fee-based design. Therefore, the age-based design is more effective in rejecting the unconfirmed transactions generated by the attacker.

4.4.2 Age-based Mempool Design

To limit attacker's chances, we propose the "Age-based Mempool Design" that addresses the limitations of our previous model. We leverage the confirmation factor or "age" of a transaction to distinguish between legitimate and malicious transactions. The age of a transaction determines how many confirmations it has achieved over time (§4.2).

For this design, we assume that the baseline size threshold of the mempool has been reached, and the mempool is only accepting transactions which are paying the relay fee as well as the mining fee. Now, for each incoming transaction, we count the number of inputs or parent transactions. We initialize a variable "average age" and set its value to 0. Next, we calculate the average age of the transaction by adding the age of each parent transaction and dividing by the total number of parent transactions. This gives an estimate of confirmation score of the incoming transaction. Then, we apply a "minimum age limit" filter on the mempool. The "minimum age limit" can take any arbitrary value greater than 0. Only if the transaction's mean age value fulfills the criteria of age limit, then the mempool accepts the transaction.

A transaction in Bitcoin has an input pointer pointing to the spendable transaction that it has previously received. In this design, we apply the check on the age of the incoming transactions. In the attack phase (§4.3.1), the spam transactions will have input pointers of a parent transaction that will not be confirmed in any block. The age of of all those parent transactions, made in the

distribution phase (§4.3.1), will be 0. Using this knowledge, we compute the average age of all the input pointers (parent transactions); minimum age value of 1 means that all transactions coming into the pool are confirmed in at least the most recent block. Once this design is implemented, if a user tries to spend his coins, he needs to have at least one valid confirmation backing up his transaction. This gives advantage to the legitimate users who typically wait for at least 6 confirmations in Bitcoin. Therefore, this design suits the legitimate users and penalizes the adversary.

Analysis of Age-based Mempool Design. Now, we analyze the working of "Age-based Mempool Design" and how it helps in countering DDoS attack. For this design, we establish that the attacker has the capability of circumventing the "Fee-based Design" and is willing to pay the relay fee and the mining fee for all transactions. Also, the attacker knows that its transactions will not be verified, so it pays higher relay and mining fee than the legitimate users.

To analyze the effectiveness of age-based countermeasures, we set a minimum age limit and a maximum age limit as thresholds for the incoming transactions. For the attacker, the only set of transactions with age value greater than 1 are generated in the distribution phase. Child transactions made in the attack phase were assigned 0 age value due to unconfirmed parent transactions. To capture that, we normally distribute the average age value of all malicious transactions from 0 to the minimum age limit. The average age value of all legitimate transactions was set from 0 to the maximum age limit. A total of 2,000 transactions were generated with half of them being malicious and half being legitimate. Then we applied the age-based design on all the incoming transactions at the mempool. We increased the age requirement for the incoming transactions and evaluated the accuracy of detection and the state of mempool for each transaction.

Evaluation Results. For evaluation of this design, we used the same confusion matrix in Table 4.1 and evaluation parameters in Table 4.1. The results in Fig. 4.3 show that upon increasing the average age the malicious transactions, (FP) decreases sharply. The mempool size decreases to a point where there are only legitimate transactions left in the mempool. Due to low (FP) and higher (TP), the precision reaches a close to 1 in Fig. 4.3(b). In Fig. 4.3(c), it can be observed that at an

average age value of 100 we achieve 60% accuracy, 80% size optimization and 80% precision. As we increase the age parameter to 200, the accuracy does not decrease as of the fee-based design, while the size ratio increases up to 90% and the precision increases up to 98%. This shows that the age-based design prevents a majority of malicious transactions from the mempool.

In these settings, if the attacker intends to spam the network, he needs to have majority of his transactions confirmed in the blockchain. However, in our attack model, we have described that confirmation is undesirable for the attacker since it results in losing budget in mining and relay fee. Moreover, using the results from Fig. 4.3(c), the attacker will have to wait a minimum of 100 blocks to relaunch the attack. With average block computation time of 10 minutes, 100 blocks lead to 16 hours of delay. Even if the attacker still plans to carry out the attack after waiting and paying all the fee, he will not be able to flood the mempool.

4.5 Summary

In this work, we identify a DDoS attack on blockchain mempools that traps users into paying higher mining fee. We note that this attack results from the application-specific policies (*i.e.* the throughput limit) in the blockchain system. In Bitcoin, this attack is more feasible due to Bitcoin's low throughput. However, in Ethereum, this attack can be costly due to high throughput. To counter the attack, we propose fee-based and age-based countermeasures and evaluate them using discrete-event simulations. Our simulation results show that the fee-based design achieves higher size optimization while the age-based design achieves a higher spam detection accuracy.

Note that in this attack, we assumed that the adversary's transactions are swiftly relayed in the network. In ideal conditions, as envisioned by Nakamoto [76], this is a fair assumption since Nakamoto assumed a completely connected P2P topology in which information propagation is fast. However, in the real world settings, if the network topology is sparse and transactions are delayed, the attack may not be highly successful. This shows that the characteristics of the P2P network influence the information propagation and the network security. Realizing that, we proceed towards the security analysis of blockchain network layer, presented in the following chapters.

CHAPTER 5: PARTITIONING ATTACKS ON THE BITCOIN NETWORK[1]

In the last two chapters, we focused on the attacks related to the application-specific policies in the blockchain systems. In this chapter, we proceed towards the network layer security problems by uncovering three forms of partitioning attacks on the Bitcoin network.

Bitcoin nodes form an overlay P2P network which is build on top of a physical network of Autonomous Systems (ASes) [76, 89]. Originally, Bitcoin was conceived as a democratic network in which nodes and the physical network did not affect the overlay network [76]. However, since 2009, the Bitcoin network has significantly scaled up and deviated from its ideal configuration. In the current network, the mining power is centralized among a few nodes, and those nodes are clustered across few ASes. We note that the centrality across ASes has a high risk of routing attacks due to the weak trust model of the "Border Gateway Protocol" (BGP). Moreover, the routing in the physical network adds block propagation delay which can lead to inconsistent blockchain views among nodes. Such inconsistencies can be exploited by malicious miners to partition the vulnerable nodes and force them to follow a counterfeit blockchain.

5.1 Contributions

In this work, we conduct a data-driven study to uncover the increasing centralization of Bitcoin nodes over the Internet and the non-uniform consensus among peers. Using that knowledge, we present spatial, temporal, and spatio-temporal partitioning attacks, summarized below.

1. We found that only 8 ASes host 30% of Bitcoin nodes and 24 ASes host 50% of Bitcoin nodes. At the organization-level, we found that only 13 organizations host 50% of the Bitcoin

[1]This content was reproduced from the following article: M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen. Partitioning Attacks on Bitcoin: Colliding Space, Time, and Logic. In *IEEE International Conference on Distributed Computing Systems*, pages 1175–1187, 2019.

nodes. Among them, only two organizations intercept 65.7% of Bitcoin hashing rate. We note that this centralization can be exploited to launch BGP attacks against the dominant ASes (**spatial partitioning attacks** §5.3.1).

2. We also observe that due to block propagation delay, the network has non-uniform consensus. Our results show that in some cases, even 5 minutes after the publication of a block, ≈62.7% of nodes do not receive the block. We show that such a behavior can be exploited to launch the **temporal partitioning attack** (§5.3.2) in which the adversary can feed false blocks to nodes and temporally partition the network.

3. To optimize spatial and temporal attacks, we explore the **spatio-temporal partitioning** attack (§5.3.3). By observing that only 5 ASes hosted ≈30% of synchronized nodes, this attack considers them as more valuable targets, thus reducing the attacker's effort.

4. We validate the partitioning attacks using the knowledge from real world settings or simulations. For each attack, we also show the attack implications on the Bitcoin network and conclude with proposed countermeasures (§5.4).

5.2 The Bitcoin Network Structure

Bitcoin consists of nodes connected in a peer-to-peer network. Upon joining the network, nodes connect to each other using public IP addresses, and use the gossip protocol to exchange network information such as transactions, blocks, and addresses. There are special nodes in the network, called *miners*, that are responsible for extending the blockchain by creating new blocks [84].

Ideally, all the participating nodes in the network need to have an updated blockchain ledger, but the growing size of the chain makes it infeasible to be used on smart devices. To address this problem, Bitcoin also uses a concept of lightweight clients or SPV clients that run on a smart device and obtain the blockchain information by connection to the full nodes. Therefore, the current Bitcoin network is structured into full nodes that are active in the main network, and lightweight nodes

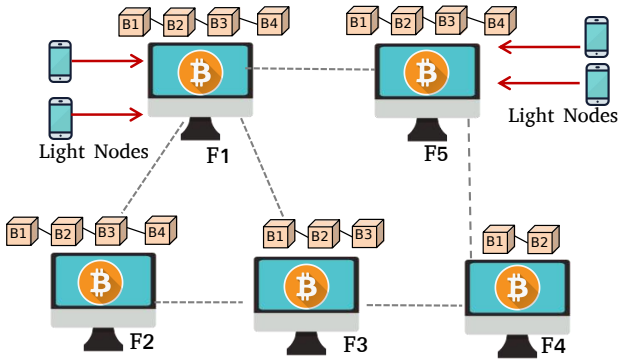


Figure 5.1: The Bitcoin network illustration showing full nodes and lightweight nodes (also called SPV clients). Lightweight nodes only have the view that their associated full nodes provide. Full nodes F1, F2, and F5 have updated views while F3 and F4 are 1-2 blocks behind.

that use services of full nodes. In Fig. 5.1, we provide an illustration of this model. For more information regarding the full nodes and the lightweight nodes, we refer the reader to [46].

5.2.1 Threat Model

In this section, we outline the basics of partitioning attacks on Bitcoin and describe our threat model. Towards that, we revisit Apostolaki *et al.*'s work [5] (referred to as the "classical attack"), providing a baseline for partitioning attacks. We highlight new targeted attacks on the network, by introducing temporal and spatio-temporal attacks, which have not be identified before.

For the spatial partitioning, we assume the adversary to be an autonomous system (AS), an ISP organization, or a nation-state. An AS hosting a fewer Bitcoin nodes can launch a BGP attack on another AS that hosts more nodes [91]. As a result, it can hijack the Bitcoin traffic, isolate the mining power, or simply harm the reputation of the target AS. For temporal attacks, we assume a malicious mining pool that attempts to fork the network and deprive an honest miner from block rewards. With soft forks, the adversary aims to create a temporary imbalance in system ramifications, such as transaction processing, and by hard forks it attempts to permanently split the network with diverging views. Additionally, due to the centralization of Bitcoin traffic and a

shift in country-level policies towards Bitcoin, we do not exclude the possibility of a nation-state adversary.

Adversarial Capabilities. In the threat model, adversaries have unique capabilities. For example, a malicious AS or organization will have the ability to announce false routing information to other ASes and separate the target AS from neighboring nodes. This, in turn, can disrupt the exchange of transactions, blocks, and mining information, thereby affecting all the network nodes.

For temporal partitioning, the adversarial mining pool will have a consistent view of the network, which will allow it to identify nodes that are behind the blockchain. *Obtaining this information is not challenging since various Bitcoin crawlers are available and can be used to access the blockchain view of nodes* [19]. This can be exploited by the malicious mining pool to identify vulnerable nodes that are one or more blocks behind. A malicious miner, for instance, can mislead those nodes by propagating false information in the network. Doing so may create a partitioning in the network, where a group of nodes are misled into following a counterfeit blockchain.

5.2.2 Data Collection

For our analysis, we crawled data from Bitnodes [19], which is a Bitcoin service supported by *Earn.com* [26]. Bitnodes maintains a persistent connection with all reachable nodes by running a full node that connects to the rest of the network. For each node, Bitnodes records useful information such as the latency, the uptime, and the latest block etc. From IP addresses, it determines the corresponding AS, organization, and location of nodes. We developed another crawler, atop Bitnodes, to acquire data and store it in our local database. We ran the crawler for two months and sampled the network snapshot at 10 minutes interval.

5.2.3 Methodology

First, we analyzed the distribution of nodes across ASes and organizations. The initial results gave us a holistic view of the network and its centralization, which we used to describe spatial partitioning attacks. Next, we analyzed the network synchronization by analyzing the blockchain view of each node. We recorded the latest block published by miners in the network and the most recent block that every node had. The difference between the two denoted how far behind the node was from the network. As shown in Fig. 5.1, nodes F3 and F4 are 1-2 blocks behind the main chain. We leveraged this information to outline temporal partitioning attacks that can be launched on Bitcoin network to isolate nodes based on their outdated view.

Measurements and Observations. Below, we discuss some key observations we made during the preliminary analysis on the Bitcoin network on February 28, 2018. The network snapshot showed that there were 13,635 full nodes in the network out which 11,382 (83.47%) nodes were up. Only 6,155 (45.14%) nodes had the most updated copy of the blockchain while 7,480 (54.86%) were 1 or more blocks behind. Among the full nodes, 12,737 (93.41%) had IPv4 address, while 579 (4.24%) had IPv6 address. The remaining 319 (2.33%) full nodes had onion addresses, meaning that they were using TOR services to run Bitcoin. During the two months data collection, the average number of nodes that were up was $\approx$10K.

5.3 Partitioning Attacks

Based on our preliminary analysis, we propose three types of Bitcoin partitioning attacks. The fundamental premise of each attack is related to the spatial positioning of nodes, the topological symmetry of the network, or the temporal consensus over the blockchain state. We define these attacks as spatial, temporal, and spatio-temporal partitioning attacks, respectively.

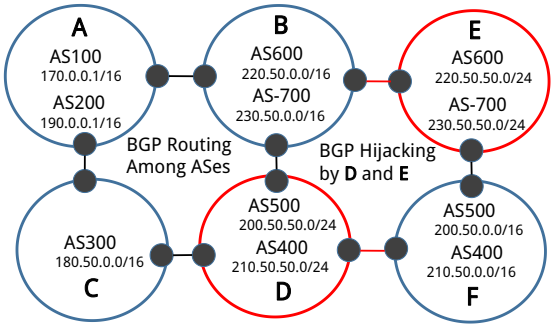


Figure 5.2: Network topology consisting of organizations, ASes and full nodes. Organizations D and E can launch BGP attacks against F and B respectively.

5.3.1 Spatial Partitioning

In this section, we analyze the centralization of full nodes and mining pools across ASes and organizations. Towards that, we revisit the prior work to evaluate the classical attack, and demonstrate that over time, the Bitcoin network has further centralized and become more vulnerable.

Attack Objectives. The objective of spatial partitioning is to isolate miners, and restricting their access to the network, or eclipsing an entire AS that hosts a large fraction of nodes. A mining pool might launch such an attack against its competitor to increase its chances to publish more blocks. A competing cryptocurrency can launch this attack to affect Bitcoin's reputation.

Attack Procedure. In Fig. 5.2, we provide an illustration of a BGP attack, which can be launched by a malicious organization or an AS. In this attack, the malicious AS announces prefixes that belong to the victim AS. As shown Fig. 5.2, organizations D and E can launch BGP attacks against organization F and B, respectively, by broadcasting more specific prefixes. Moreover, the attack can be made more targeted by announcing prefixes addressing only Bitcoin nodes. This attack relies on two major factors: the total number of ASes and organizations, and the total number of nodes hosted in each of them. In particular, if the total number of ASes and organizations hosting full nodes is large, the attack becomes costly. Similarly, if the number of nodes is concentrated within a few ASes, that makes a better target rather than attacking arbitrary ASes with fewer

Table 5.1: Top 10 ASes and Organizations that host Bitcoin nodes as of February 28th 2018. Note that the network is more centralized with respect to organizations than ASes, and AS24940 host the maximum number of Bitcoin nodes.

ASes	# of Nodes	Total Nodes %	Organizations	# of Nodes	Total Nodes %
AS24940	1,030	7.54%	Hetzner Online GmbH	1,030	7.54%
AS16276	697	5.11%	Amazon.com, Inc	756	5.54%
AS37963	640	4.69%	OVH SAS	700	5.13%
AS16509	609	4.47%	Hangzhou Alibaba	640	4.69%
AS14061	460	3.37%	DigitalOcean, LLC	503	3.69%
AS7922	414	3.04%	Comcast Communication	414	3.04%
AS4134	394	2.89%	No.31, Jin-rong Street	394	2.89%
AS51167	288	2.11%	Contabo GmbH	288	2.11%
AS45102	279	2.05%	Alibaba (China)	279	2.05%

nodes. To evaluate that, we carried out two experiments to observe the total number of ASes hosting Bitcoin nodes and the distribution of nodes among those ASes.

Practical Considerations. Our results show that the full nodes in Bitcoin are highly centralized at the AS and organization level. Compared to [5], the network has become even more centralized, and more vulnerable to BGP hijacking and routing attacks. In particular, we observed that among the total of 84,903 ASes in the world [85], only 8 (0.0094%) ASes host 30% Bitcoin nodes. 24 (0.028%) ASes host 50% while 1,660 (1.95%) ASes host 100% Bitcoin nodes. This shows a significant difference in the number of ASes that host 50% and 100% full nodes. To understand that, we plot CDF of ASes that host the traffic of full nodes in Fig. 5.3.

Similarly, we observed that the top 8 organizations intercepted 30% Bitcoin traffic and the top 13 organizations intercepted 50% traffic, collectively. We also noticed that each organization controlled one or more ASes, alluding to the possibility of a fine-grained partitioning attack.

In Table 5.1, we show the top 10 ASes and organizations along with the percentage of total nodes that they host. AS24940 hosts 7.54% nodes and its corresponding organization *Hetzner Online* also hosts 7.54% nodes, meaning that the Bitcoin traffic routed by Hetzner Online entirely goes through AS24940. On the other hand, Amazon.com routes 5.54% of the traffic while AS16276 intercepts 5.11% traffic. This shows that Amazon.com owns another AS besides AS16276 that

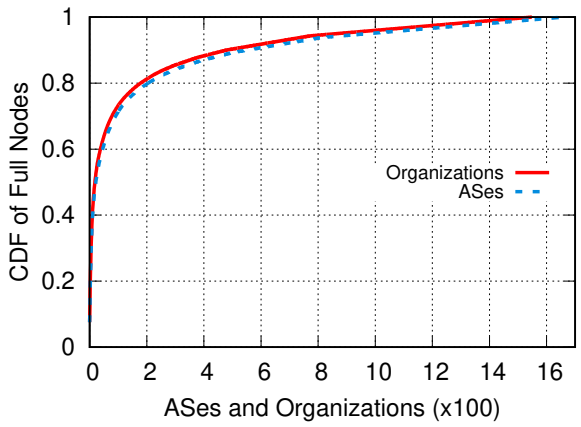


Figure 5.3: CDF of the Bitcoin full nodes in ASes and organizations.

Table 5.2: Top 5 mining pools per hash rate, ASes, and organizations. 65.7% mining data goes through only three organizations. Alibaba intercepts at least 60% of the mining data. We exclude the remaining 12 mining pools from the study as their contribution to the hash rate is minimal.

Mining Pool	**H. Rate %**	**ASes**	**Organizations**
BTC.com	25%	AS37963 AS45102	Hangzhou Alibaba AliBaba (China)
Antpool	12.4%	AS45102	AliBaba (China)
ViaBTC	11.7%	AS45102	AliBaba (China)
BTC.TOP	10.3%	AS45102	AliBaba (China)
F2Pool	6.3%	AS45102 AS58563	AliBaba (China) Chinanet Hubei
12 others	34.3%	—	—

also routes traffic. This model is similar to the illustration shown in Fig. 5.2.

Mining pools are another important part of Bitcoin, since they are responsible for extending the blockchain and maintaining its state. Mining pools consist of miners on the Internet communicating via a special mining protocol known as the "Stratum Mining Protocol" [21]. All miners compute PoW and send the result to the stratum server address specified by the mining pool. The stratum address is made public by the mining pool. As such, if the link to the stratum server is compromised, the mining pool gets disconnected and its aggregate hash rate decreases. To analyze the distribution of stratum servers, we carried out two experiments. First, we gathered informa-

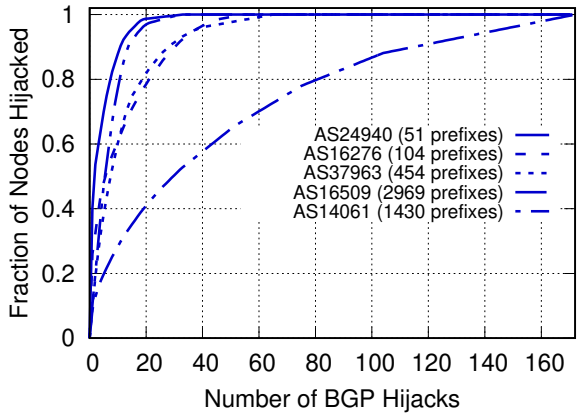


Figure 5.4: CDF of top 5 ASes vulnerable to BGP attacks. The key shows total BGP prefixes announced by AS. For 8 ASes, 80% nodes can be isolated by hijacking 20 BGP prefixes.

tion about major mining pools in Bitcoin and their hash rate from *Blockchain.info* [10]; results are reported in Table 5.2. Next we selected the top 5 mining pools, which had an aggregate hash rate of 65% of the total in the Bitcoin network. We then collected the stratum address of the selected mining pools from their websites and traced the IP address corresponding to each stratum address [12, 4, 40]. We mapped each IP address to the AS hosting the stratum server. We found that 3 ASes had 65% of Bitcoin mining pool traffic while one organization "AliBaba" alone had more than 50% of the Bitcoin mining pool traffic. We report our results in Table 5.2. In the light of our threat model, and given an adversary capable of BGP hijacking, policy enforcement at an organization level, or collusion, having an organization hosting more 50% of the mining power makes such an attack even more effective.

Attack Validation and Implications. In this section, we validate our hypothesis regarding BGP hijacking on Bitcoin ASes and organizations. BGP routing attacks on Internet happen frequently. In 2008, a service provider from Pakistan hijacked Youtube traffic by announcing more specific BGP prefixes than the ones announced by Youtube [48]. Similarly, in 2014, a Canadian ISP hijacked prefixes of 19 organizations hosting Bitcoin traffic including Amazon, Digital Ocean, and Alibaba [50]. In 2017 alone, 14,000 BGP attacks were launched against major ASes [86].

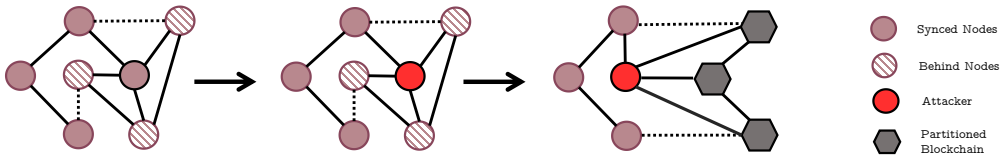


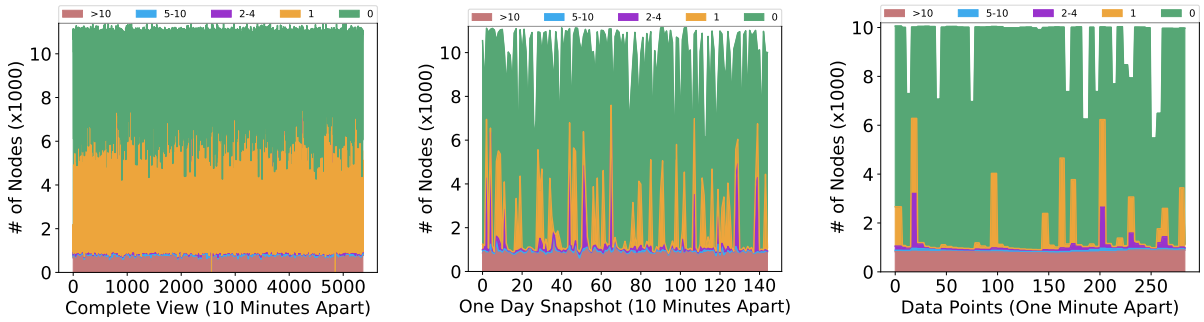
Figure 5.5: An illustration of the temporal attack. The attacker establishes connections with nodes and identifies vulnerable nodes that have an outdated view. Vulnerable nodes have have not been provided new blocks by surrounding peers, which shows their weak relationship/connectivity. We annotate this weak relationship with dotted lines. The attacker feeds his copy of blocks to vulnerable nodes, thereby partitioning the network into two conflicting chains.

To validate the attack and its impact, we selected the top 5 ASes from Table 5.1, and enumerated the IP addresses of full nodes hosted by these ASes. Next, we grouped the IP addresses based on the BGP prefixes announced by each AS and calculated the number of BGP prefixes required to isolate a percentage of full nodes. We report results in Fig. 5.4, showing that except for AS16509, 95% of full nodes in all other ASes are vulnerable, once fewer than 40 BGP prefixes are hijacked.

Spatial partitioning is detrimental to the Bitcoin network as it facilitates other major attacks including eclipse attacks and the 51% attack. As shown in Table 5.2, if an attacker hijacks 3 ASes, he can isolate more than 60% of the Bitcoin hash power. As Fig. 5.4 shows that by hijacking 15 BGP prefixes, the attacker can cut 95% traffic of AS24940 that hosts 1,030 full nodes. By isolating the hash power, an attacker can cause delays in the block creation and the transaction confirmation.

5.3.2 Temporal Partitioning

Temporal partitioning involves isolating nodes that are a few blocks behind the rest of the network. As shown in Fig. 5.1, three nodes have the most updated copy of the blockchain, while nodes F3 and F4 are 1–2 blocks behind. These nodes might be behind the main chain due to a number of reasons, such as the network latency due to increasing network size or malicious peer behavior. Therefore, these nodes have an outdated blockchain view and remain vulnerable to partitioning attacks. Fig. 5.5, provides an illustration of this attack.



(a) General trend in the network (b) One day snapshot (c) One minute sampling

Figure 5.6: Temporal consensus in Bitcoin network. Y-axis denotes number of nodes in 1000. In each figure, green region denotes the up-to-date blocks. Yellow region denotes 1 block behind. Purple, blue, and magenta regions represent nodes that are 2–4, 5–10, and $\geq$ 10 blocks behind respectively. Fig. 5.6(a) shows the overall network, Fig. 5.6(b), shows a day (March 25) that offers greater attack opportunity, and Fig. 5.6(c) shows consensus pruning during 10 minutes.

Attack Objectives. The objective of the temporal partitioning is the isolation and subversion of nodes or a group of nodes within the network. Latency in updating the blockchain is a well known vulnerability of Bitcoin, which is confirmed in our data. Propagation delays are known to be key contributors towards the latency [34]. Propagation delay is influenced by the number of hops between nodes due to sparse peering, and the time required by software clients to verify and forward a block. Solutions have been proposed that cluster nodes to reduce latency [98, 41], but the authors note this may increase the potential for partitioning attacks. This indicates a trade-off between spatial and temporal vulnerability.

Attack Procedure. Our analysis shows that several times a day, a significant fraction of nodes are not up-to-date. We report our findings in Fig. 5.6 where the x-axis denotes a time-index for network observations (one observation every 10 minutes in Fig. 5.6(a) and Fig. 5.6(b), and one every minute in Fig. 5.6(c)). The y-axis is stacked, meaning that curves are cumulative. The green part shows the synchronized nodes, the yellow part shows nodes that are 1 block behind. The description of remaining colors is in the figure.

From Fig. 5.6(a), we were able to make following observations. (1) Generally, a majority of nodes

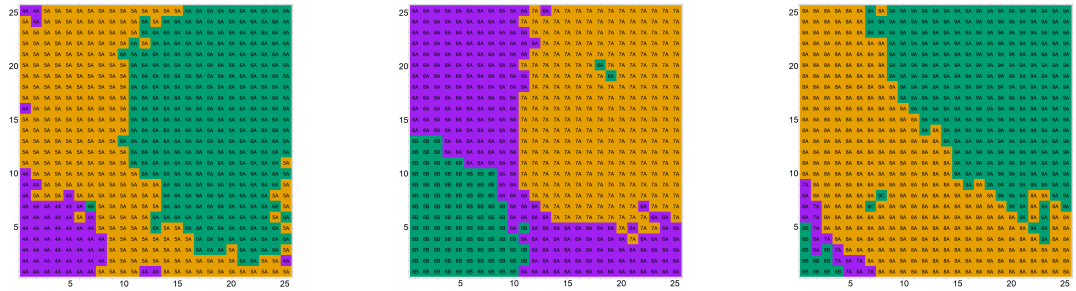
($\approx 50\%$) remains synchronized on the blockchain state. These nodes do not lag behind in the main chain for a long duration. (2) 10% nodes are forever behind the main blockchain. They do not update their blockchain and as such, they have no benefit in the network. (3) 30-40% nodes in Bitcoin occasionally waver in terms of their view of the blockchain. Possibly due to network latency or consensus delay, they lag behind the most recent block.

Focusing on a single day shown in Fig. 5.6(b), we observed that some yellow and purple spikes are larger and wider than others. From Fig. 5.6(b), we made the following observations. (1) There is non-uniform consensus in the network. (2) The most frequent delay among the blocks is 1 block indicated by yellow region, followed 2-4 blocks, indicated by the purple region. (3) On various occasions, yellow and purple spikes can reach up to 7,000 nodes; approximately 90% of the network can be partitioned if an attacker isolates them.

It is surprising to note the deteriorating network synchronization in the Bitcoin network. In 2013, Decker *et al.* [34] noted that a Bitcoin block is delivered to all nodes in less than 12 seconds. In contrast, our measurements present a different and more concerning picture. One possible reason is that since 2013, the Bitcoin network size has grown from ≈3.5K nodes to more than 10K nodes. We believe that an increasing network size and the constant network outdegree (8) is the main reason for poor network synchronization .

In Bitcoin, on average, a block is published every 10 minutes. In the previous two experiments, we observed that with fine grained sampling, on a given day, the attacker can isolate a group of nodes that are behind the main chain. To further analyze this behavior, we performed another experiment that involved per-minute sampling of network. Our objective was to observe the consensus distribution among peers immediately after a block broadcast. We plot the results obtained from the third experiment in Fig. 5.6(c) showing that there are vulnerable spots in the network in which up to 90% of the network is 1-4 blocks behind. As such, the non-uniform consensus presents an opportunity whereby an attacker can find a time window to isolate a group of targeted nodes.

Simulation and Attack Validation. To validate the insights obtained from our data, we developed



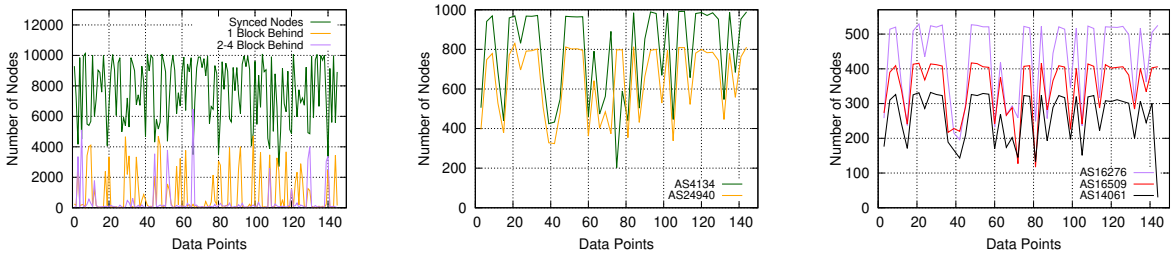
(a) Time Step 151 (b) Time Step 201 (c) Time Step 251

Figure 5.7: Simulation of temporal attack. Fig. 5.7(a) shows fork B emerging at node [7,7]. Compare the color distribution to the peaks of Fig. 5.6(c) above. Two blocks later in Fig. 5.7(b) fork B has control of 1/6 of the nodes. In Fig. 5.7(c) the longer chain A overwhelms fork B but has lost synchronization so cannot prevent emergence of a new fork C.

a simulation model in R to test the temporal partitioning attack. The simulator was tested in base simulation scenarios, such as zero and perfect communication among nodes. As an internal error check, and to make the simulation more realistic, each simulated node maintains a 64-bit MD5 hash linked chain of values updated to its current fork. By adjusting parameters, the simulation was capable of accurately representing the state of the network as we observed in our dataset.

Fig. 5.7 shows a sample of results obtained from simulation, where the attacker has 30% of the network hash rate. Once a portion of the network is isolated, it can be sustained with successive forks, since the isolated nodes naturally assume that delay is due to network issues. As such, they do not know that new blocks are taking more time to calculate due to the lower hash rate of the attacker. Meanwhile, the main chain loses its hash rate and is therefore, less capable of responding. Note that the cost of launching a temporal attack is much less than the spatial attack, provided that the attacker has the consistent view of the network as shown in Fig. 5.6.

Implications. Even a short term fork in the network would cause sufficient disruption to invalidate transactions. Such an attack is likely to result in significant loss to network stakeholders. Quantifying the impact of adverse events on Bitcoin has been inconclusive [42][36], and is dependent upon user perception [81]. However, once the targeted nodes are isolated, as shown in Fig. 5.5,



(a) One day snapshot (b) Top 1-2 synced nodes ASes (c) Top 3-5 synced nodes ASes

Figure 5.8: Spatial and temporal distribution of nodes for the day defined in Fig. 5.6(b). For the synced nodes in Fig. 5.8(a), we outline their distribution across top five ASes in Fig. 5.8(b) and Fig. 5.8(c). On average, AS4134 hosts most of the nodes.

the soft fork will create a temporary partition in the network. The isolated nodes will be following a counterfeit blockchain with different transactions from the main chain. Therefore, when nodes recover from the fork, the attacker's blocks will be rejected, and all transactions belonging to legitimate users in those blocks will also be reversed. This will require a major update on the set of all UTXO's at each node, and a system-wide check on the transactions being reversed.

5.3.3 Spatio-temporal Partitioning

We now analyze how an attacker can leverage spatio-temporal patterns to find vulnerable spots in the network, through which he can effectively isolate a group of nodes. From our data analysis, we found the feasibility and cost of this attack compared to spatial and temporal partitioning.

Attack Objectives. In this attack, the adversary aims to split the network based on the network's vulnerability to both the spatial and temporal partitioning. As shown in Fig. 5.6(a) and Fig. 5.6(b), the purple and yellow nodes are vulnerable to temporal attacks. However, the attacker cannot launch the same attack on nodes in the green region since they are up-to-date. These nodes can still be partitioned based on the BGP attack presented in spatial partitioning. A combined effect of both attacks will be an optimized and targeted attack that can prolong the partitioning effects.

Note that for a BGP attack on nodes within the green region, the attacker does not have to isolate

Table 5.3: Top 5 ASes that hosted all the synchronized nodes in Fig. 5.6(b) for 24 hours.

AS	Organization	Nodes	Percentage
AS4134	No.31, Jin-rong	993	9.57%
AS24940	Hetzner Online	830	7.98%
AS16276	OVH SAS	530	5.22%
AS16509	Amazon.com	417	4.19%
AS14061	DigitalOcean	332	3.23%

all target nodes. Since these up-to-date nodes are connected with each other, therefore, an attack on a subset of nodes can have a cascade effect, thereby compromising all other nodes.

Attack Procedure and Validation. For a successful attack, the attacker needs information about the ASes and organizations of the synced nodes as well as nodes that are behind. The feasibility of this attack depends on the adversary's capabilities. Per our threat model, if the attacker is an AS, it will prefer to hijack BGP prefixes to damage Bitcoin. As such, it will prefer maximum nodes in the green region and minimum nodes in yellow and purple region, to maximize the attack severity. If the attacker is a mining pool, then it will launch a temporal attack, and will prefer minimum nodes in green region and maximum nodes in other regions. However, if the attacker is a cloud service provider that has both routing and mining capabilities, then it can launch both spatial and temporal attacks. Therefore, this attack is adjustable to the capabilities of an attacker.

Although multiple attack scenarios and case studies can be drawn for spatio-temporal partitioning but in the interest of space, we illustrate one case study. From our simulations, we observed that the temporal partitioning forks the network at a faster rate than spatial attacks. Therefore, we assume a case in which cloud provider waits for minimum number of synced nodes, and launches a spatio-temporal attack. In Table 5.3, we enlist the top 5 ASes and organizations that hosted most synchronized nodes in Fig. 5.8(a). We observed that 28% of synced nodes are hosted within the top 5 ASes. We plot their hosting pattern over a full day in Fig. 5.8(b) and Fig. 5.8(c). The cloud provider can spatially attack the synchronized nodes by hijacking five ASes and temporally attack the remaining nodes that are one or more blocks behind. This can eventually lead to a hard fork.

Implications. Spatio-temporal attack is an optimized and targeted attack that provides multiple attack opportunities to a strong adversary to take down the network with minimal effort. As demonstrated by our results in Fig. 5.8, at a given time, more than 50% of nodes can be behind the main blockchain and vulnerable to temporal attacks. Moreover, at the same time, the remaining synced nodes can be attacked by hijacking BGP prefixes of their hosting ASes and organizations. The attacker can select a suitable trade-off between the lagging nodes and synced nodes, based on the attacker's capabilities, and disrupt the network. For a successful attack on synced nodes, the attacker may just have to isolate a small number of nodes that relay blocks to each other, and due to the cascade effect, remaining nodes will eventually collapse. Note that the synchronized nodes can help other nodes to recover from the temporal partitioning attacks. However, in the spatio-temporal attack, that recovery will not be possible leading to a hard fork.

5.4 Countermeasures

In this section, we discuss the preventive measures to counter the partitioning attacks. To prevent spatial partitioning, mining pools should spread stratum servers across various ASes. This can resist the centralization of stratum servers and raise the attack cost, since the attacker will have to hijack more BGP prefixes to isolate the targeted pool. Furthermore, large Bitcoin exchanges such as Coinbase and Bitstamp should also host their full nodes across multiple ASes to prevent spatial attacks. In Bitcoin, spatial partitioning is an artifact of BGP hijacking and to counter that, Zhang *et al.* [114] propose reactive and proactive defense strategies that are based on the idea of "bogus route purging and valid route promotion" that can prevent BGP attacks on ASes across the Internet.

Temporal partitioning results from malicious peer behavior towards nodes that are behind the main chain. Although nodes can be behind due to various factors, the absence of a trusted central authority, makes them unaware of their condition. To counter that, a node can compare the timestamp of its latest block and the expected time for the next block. If a node does not receive a block on the expected time, it can try new outbound connections to increase its network reachability.

5.5 Summary

In this work, we uncover three forms of partitioning attacks on the Bitcoin network by measuring and characterizing the increasing centralization of the Bitcoin nodes across ASes and decreasing network synchronization. We call them spatial, temporal, and spatio-temporal attacks and show their impact on the Bitcoin network through measurements and simulations. The spatial partitioning attack exploits the centrality of Bitcoin nodes across ASes to optimal BGP attack opportunities. The temporal partitioning attack exploits the weak network synchronization to allow a mining pool to create soft forks in the network. Finally, the spatio-temporal partitioning attack exploits the commonalities between spatial and temporal attack vectors to create long-term partitioning.

Standing out in our work is the the temporal partitioning attack which is a novel contribution to the blockchain security. We are the first to note the decreasing network synchronization in the Bitcoin network. We believe that this is due to the increasing network size and block propagation delay. However, to confirm that, we conduct a root cause analysis, presented in the next chapter.

CHAPTER 6: ROOT CAUSE ANALYSIS FOR BITCOIN NETWORK SYNCHRONIZATION

In this chapter, we conduct a root cause analysis to explore the key factors behind the decreasing network synchronization in the Bitcoin network. Network synchronization ensures that all nodes have an up-to-date and consistent view of the blockchain, whereas the lack of such a view, i.e., nodes being behind the current state of the blockchain, makes the Bitcoin network vulnerable to the partitioning attacks [92]. In Bitcoin, a peer-to-peer (P2P) network of more than 10,000 nodes [72], synchronization is affected by the block propagation delay. Recent work has shown that the synchronization in the Bitcoin network deteriorated in 2018 [92], when compared to 2013 [34], due to the increasing network size: between 2013 and 2018, the number of *reachable* nodes increased from $\approx$3K nodes to $\approx$10K nodes, thereby increasing the block propagation delay.

The fundamental reason why the block propagation delay increases as a function of the Bitcoin network size is twofold. First, in the current design of the Bitcoin protocol, each *reachable* node in the Bitcoin network can connect to only eight other *reachable* nodes, making that the outdegree of each node in the network. As a result, increasing the network size while fixing the outdegree will only mean that longer paths will connect any two nodes in the network on average, and a block has to traverse more hops to reach its destination, eventually negatively affecting the Bitcoin network synchronization by increasing the block propagation delay. Second, the increasing number of Bitcoin nodes are possibly hosted across different Autonomous Systems (ASes) [35, 5], with various hosting patterns. Given the spatial distribution of the nodes across those ASes, and the complexity (length and policy) of the routes between those ASes, the increase in the number of nodes will also increase the block propagation delay.

Given the stability of the latency distribution on the Internet recently, one would anticipate the overall Bitcoin network synchronization will stay the same in the absence of any significant change to the Bitcoin protocol, and given a constant network size. Surprisingly, however, recent measure-

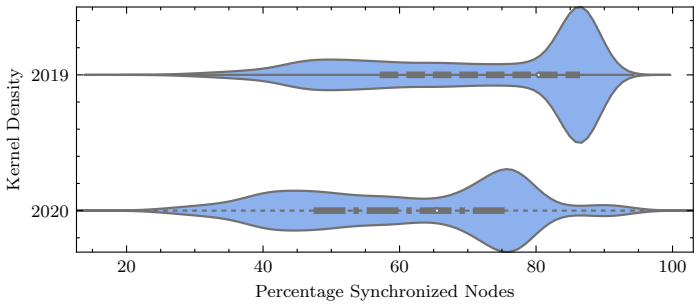


Figure 6.1: Bitcoin network synchronization in 2019 and 2020. Synchronization is determined by the percentage of nodes with the up-to-date blockchain. In 2019, the mean and median network synchronization were 72.02% and 80.38%, respectively. In 2020, the mean and median network synchronization decreased to 61.91% and 65.47%, respectively. The kernel density shape also shows that the Bitcoin network synchronization decreased in 2020.

ments show that, while the *reachable* network size has remained constant (≈10K nodes), the network synchronization has deteriorated even further, as shown in Fig. 6.1, which captures a kernel density plot of the network synchronization in 2019 and 2020.

6.1 Background and Motivation

To investigate what caused such changes, in both cases, we collected the Bitcoin network data from Bitnodes [23] and measured the synchronization. Through analysis, we found that the number of *reachable* nodes between September and December 2019 was ≈10K, where the average synchronized nodes' percentage was ≈72%. However, we found that while the average number of *reachable* nodes stayed the same between January and April 2020, at ≈10K, the average percentage of synchronized nodes decreased to only ≈62%.

The decreasing network synchronization is alarming, suggesting that the Bitcoin network is becoming more vulnerable to partitioning attacks [92]. Moreover, these results suggest that the network synchronization cannot be only attributed to the network size, and there must be other hidden factors that affect it, which warrants further exploration through root cause analysis. To this end, in this work, we explore four plausible factors that may influence the block propagation and network

synchronization, which we briefly discuss in the following.

Unreachable Network. There are two types of Bitcoin nodes, *reachable* and *unreachable*. The *reachable* nodes establish outgoing connections with and accept incoming connections from other nodes. The *unreachable* nodes (often behind a NAT [35]) only establish outgoing connections. In the prior work, block propagation was measured by observing the interactions among *reachable* nodes [23, 92, 78], while the growth of *unreachable* nodes and their impact on network synchronization is not characterized yet. While the *reachable* network size remains unchanged, as shown in Fig. 6.1, the *unreachable* network size might be growing and may influence the block propagation. Therefore, the first part of our analysis includes a mapping of the *unreachable* network and determining its impact on the Bitcoin network synchronization.

Addressing Protocols. When outbound connections of a node are dropped, the node tries to establish new connections with other nodes until the outbound slots are filled. While making those connections, the node does not distinguish between *reachable* and *unreachable* IP addresses. Ideally, a node's IP address database, `addrMan`, should contain only *reachable* addresses so that each outgoing connection is successful. In contrast, if `addrMan` is dominated by *unreachable* IP addresses, the node wastes time in failed connection attempts and, as a result, might not immediately receive a block if the outgoing slots are not filled. The second part of our analysis is by studying the Bitcoin addressing protocol and empirically evaluating the success rate of outgoing connections establishment. We conjecture a low outgoing connection establishment success rate due to the addressing protocol will contribute to the deteriorating network synchronization.

Relaying Protocols. The Bitcoin ideal design [76, 44] assumes that each node "broadcasts" blocks to the entire network in *lock-step* synchronous [44, 83] manner, meaning that the block is concurrently released to all connections. Exploring how the Bitcoin Core implements the broadcast mechanisms is essential, although not done before. Therefore, the third part of our analysis is by studying the Bitcoin block relaying protocol and analyzing its impact on network synchronization.

Network Churn. The Bitcoin network is *permissionless* and nodes can leave the network at any

time. The nodes churn decreases the average node outdegree and network synchronization. Moreover, when new nodes replace the synchronized nodes, it could take the new nodes several days to download the blockchain. Only after downloading the blockchain they will be able to propagate newly mined blocks. Therefore, a high churn is undesirable. The fourth part of our analysis is measuring and characterizing the network churn and how it affects synchronization.

Contributions. We pursue a measurement-based approach for evaluating these factors. First, we set up a data collection system that connects to all *reachable* nodes and collects IP addresses of *unreachable* nodes. Our longitudinal analysis captures the number of *reachable* and *unreachable* nodes, and characterizes the network churn. Our source code inspection unveils characteristics of the the block relaying protocol and the network addressing protocol. We conduct experiments to evaluate those factors' impact on block propagation. Our contributions are as follows:

1. We conduct a comprehensive mapping of the Bitcoin network. As a result, we discover ≈29K *reachable* and ≈694K *unreachable* nodes, respectively. By probing the ≈694K *unreachable* nodes, we find that ≈54K *unreachable* nodes are active at any time (§6.3.1).

2. Through source code analysis, we unveil a major limitation in the Bitcoin addressing protocol. We note that the addressing protocol does not distinguish between *reachable* and *unreachable* nodes, thus increasing the failure rate of the outgoing connections. Our experiments on a custom Bitcoin node showed a failure rate of 88.8% (§6.3.2).

3. We study the *in situ* block relaying protocol and measure its impact on block propagation. We found that Bitcoin implements a round-robin block relaying for each connection, rather than the simultaneous block broadcast. Our experiments show that the round-robin relaying can add up to 17 seconds of delay in relaying block to the last connection (§6.3.3).

4. We measure the impact of nodes churn on synchronization. Our analysis reveals that *reachable* nodes have a short life. Each day, ≈8% *reachable* nodes leave the network, replaced by an equal number of new *reachable* nodes. Compared to 2019, the churn among the synchronized *reachable* nodes has doubled in 2020 (§6.3.4).

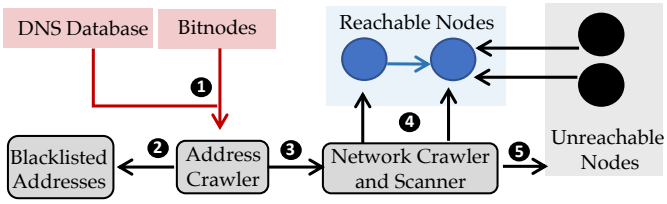


Figure 6.2: Data collection workflow. The "Address Crawler" collected IP addresses of *reachable* nodes from DNS database and Bitnodes, removed blacklisted addresses, and forwarded them to the "Network Crawler and Scanner" which operated our Bitcoin node that sent GETADDR messages. After collecting IP addresses of *unreachable* nodes, it sent them a `VER` message using *Scapy*. *Unreachable* nodes that responded to the `VER` message were labeled as *responsive* nodes.

6.2 Data Collection Methodology and Overview

Our analysis is based on the data we have collected from the Bitcoin network in a duration of 60 days (04 April, 2020 to 04 June, 2020). For this purpose, we have implemented our data collection system as shown in Fig. 6.2. In this section, we present our data collection methodology, step-by-step, and an overview of our dataset.

6.2.1 Collecting Reachable Bitcoin Node Addresses

As a first step of our data collection, we collect all reachable addresses in the Bitcoin network. Since our data collection system relies on some key designs of Bitcoin network protocol, we first present some of such details before discussing our data collection methodology.

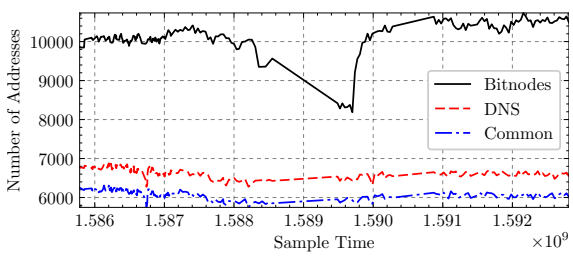
Default Connection Limits. By default, in Bitcoin, a *reachable* node can establish 8 outgoing and 117 incoming connections, while an *unreachable* node can only establish 8 outgoing connections [18]. Since, *unreachable* nodes drop incoming connections, therefore, no two *unreachable* nodes can directly connect to each other. As such, information exchange (*i.e.* transactions or blocks) between two *unreachable* nodes is enabled by *reachable* nodes.

Node Bootstrapping. When a node joins the Bitcoin network for the first time, it queries nine DNS seeders that are hard-coded in the *chainparams.cpp* file [54]. The DNS seeders return a list of IP addresses to which the node may establish outgoing connections. Once a connection is established, the node sends a GETADDR request to each connected node and receives an ADDR message in response. The ADDR message contains up to 1000 IP addresses that the sending node selects from its `addrMan` database. If the `addrMan` database has less than 1000 IP addresses, the sending node sends all those addresses in the ADDR message. In each ADDR message, a node also sends its own IP address with the current UNIX timestamp.

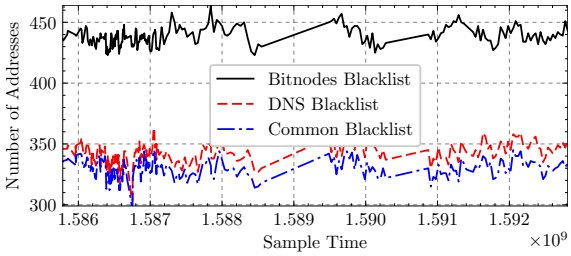
Upon receiving the ADDR message, a node stores IP addresses in either a `tried` table or a `new` table. The `tried` table stores addresses that the node has connected to in the past, while the `new` table stores addresses that the node sees for the first time. To establish a new connection, the node randomly selects an address from `new` or `tried` table with an equal probability. If a successful connection is established to an address from the `new` table, it is moved to the `tried` table.

Design of our Collection System. Given these characteristics of a Bitcoin node, we set up our data collection system to connect to the *reachable* nodes and discover the *unreachable* nodes. Our system is shown in Fig. 6.2, where we first collected the IP addresses of *reachable* nodes from Bitnodes and a DNS server database. Bitnodes has been extensively used in prior works [92, 5] to sample *reachable* nodes, and we adopt the same approach in our study. Additionally, we gained access to a Bitcoin DNS server database maintained by Luke Dashjr who has hard-coded his DNS server address in Bitcoin Core [60]. Luke Dashjr's DNS database records IP addresses of nodes that queried his DNS server. The objective of using the DNS database was to connect with the *reachable* addresses that may be skipped by Bitnodes, thus ensuring a full network coverage.

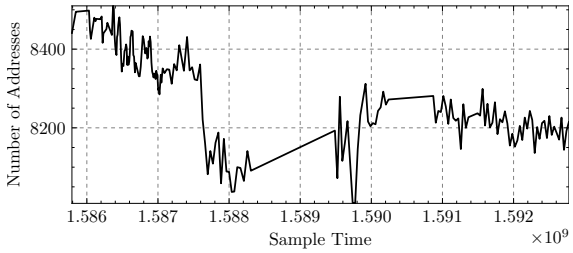
Ethical Considerations. During this study, we were advised to avoid connecting to any node hosted in the national critical infrastructure sector (*i.e.* military infrastructure). In compliance, we compiled a list of 4 million IP addresses that belong to the critical infrastructure. As shown in Fig. 6.2, after collecting IP addresses from Bitnodes and the DNS database, we removed the IP



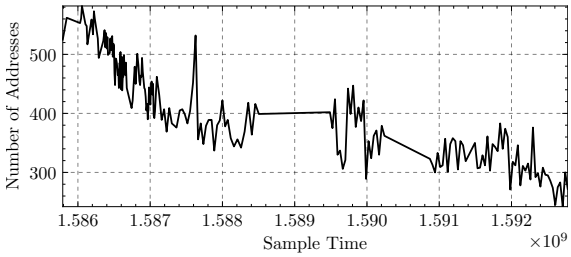
(a) The number of IP addresses in Bitnodes, DNS, and common among the two.



(b) The number of banned IP addresses in Bitnodes, DNS, and common among the two.



(c) The number of *reachable* nodes we connected.



(d) The number of *Reachable* nodes not in Bitnodes

Figure 6.3: Preliminary experiment results. On average, from Bitnodes and DNS server database, we collected 10,114 and 6,637 IP addresses, respectively. Among them, 439 and 342 addresses belonged to the critical infrastructure. Our Bitcoin node connected with 8,270 nodes on average.

addresses that mapped to the critical infrastructure sector and excluded them from analysis.

After removing these addresses, we provide the remaining set of *reachable* IP addresses to the "Network Crawler and Scanner" (Fig. 6.2) that consists of a Bitcoin node equipped with a packet generation tool called *Scapy*. The Bitcoin node connects with all the *reachable* nodes and exchanges ADDR messages to collect IP addresses of *unreachable* nodes. It then uses *Scapy* to probe the *unreachable* nodes and mark the ones that responds to the VER message.

Overview of Reachable Addresses. In Fig. 6.3, we report (1) the number of IP addresses collected from Bitnodes and DNS server Fig. 6.3(a), (2) the number of excluded IP addresses in Bitnodes and DNS server Fig. 6.3(b), (3) the number *reachable* nodes with which we connected Fig. 6.3(c), and (4) the number of *reachable* nodes, skipped by Bitnodes Fig. 6.3(d). On average, Bitnodes provided 10,114 IP addresses out of which, 439 were excluded. The DNS server database provided 6,637 IP addresses out of which 342 were excluded. 6,078 IP addresses were common in both

Bitnodes and DNS database out of which, 329 were excluded.

In a duration of 60 days, our Bitcoin node connected with 28,781 unique *reachable* IP addresses. In each experiment, our node connected with 8270 *reachable* nodes on average, where 95.78% of all nodes used the default 8333 port while the remaining used 264 unique ports, other than 8333. Moreover, some *reachable* IP addresses provided by the DNS server database were not present in Bitnodes. As shown in Fig. 6.3(d), on average, we connected with 404 IP addresses that were not present in the Bitnodes dataset. This shows that using the DNS server database for experiments was useful in extending our coverage of the *reachable* network.

6.2.2 Collecting Unreachable Addresses

After connecting to the *reachable* nodes, our node sent GETADDR requests to all reachable nodes. In response, the nodes replied with ADDR message containing up to 1000 IP addresses selected from their `new` and `tried` tables [54]. In algorithm 1, we provide the methodology of collecting *unreachable* addresses from the *reachable* nodes. For simplicity, we define N_r and N_u as two sets containing IP addresses of *reachable* and *unreachable* nodes, respectively. We further define P_i as an address in N_r to which our Bitcoin node sent GETADDR request. If the ADDR message from P_i contained one unique address that was not sent in prior ADDR messages, our node repeated the GETADDR request. If a new message contained all IP addresses that were sent in previous ADDR message, we stopped sending GETADDR messages and assumed that the node had sent all addresses from its tables. Through iterative requests, we collected IP addresses from each node's `new` and `tried` tables. For all addresses received, our node filtered *reachable* addresses from Bitnodes and DNS server database to obtain the *unreachable* addresses.

6.2.3 Discovering Responsive Unreachable Addresses

Once the network crawler and scanner (Fig. 6.2) obtain the list of all *unreachable* addresses, it scanned them to detect *responsive* nodes. In this work, the term *responsive* nodes refers to *un-*

Algorithm 1: Discovering *unreachable* IP addresses.

```
Input: reachable IP addresses in N_r
Initialize Empty list of unreachable IP addresses N_u
foreach P_i ∈ N_r do
    Send GETADDR and Receive ADDR messages
    if ip addresses ∈ ADDR ∉ N_r or N_u then
        foreach ip address ∉ N_r do
            N_u ← ip address
        repeat
    else
        foreach ip address ∉ N_r do
            N_u ← ip address
        continue
Output: IP addresses of unreachable nodes in N_u
```

Algorithm 2: Discovering *responsive* addresses in N_u.

```
Input: reachable IP addresses in N_u
Initialize List of responsive nodes N_3
foreach P_i ∈ N_u do
    Send VER message
    if P_i responds to VER then
        N_3 ← P_i
    else
        mark P_i as silent
Output: IP addresses responsive nodes in N_3
```

reachable nodes that drop the incoming connection by responding to the VER message. As a result, despite the node being *unreachable*, we know that it is running Bitcoin.

To verify our method, we deployed three *unreachable* nodes inside our network and sent the a VER message through our *reachable* node outside of our network. We observed that all three *unreachable* nodes dropped our connection by responding to the VER message with FIN flag set to 1. We applied this methodology to all the *unreachable* IP addresses received from the *reachable* nodes. If the *unreachable* address responded to VER message with FIN flag set to 1, we marked the address as *responsive*. algorithm 1 outlines our methodology of detecting *responsive* nodes.

Considering the high volume of *unreachable* addresses, we manually crafted Bitcoin VER message in *Scapy* and applied algorithm 2. The script was deployed on a commodity computer which sent 250 parallel requests to the *unreachable* addresses. This is to be noted that the procedure of using VER message is a heuristic one that may only work for nodes that allow such incoming requests

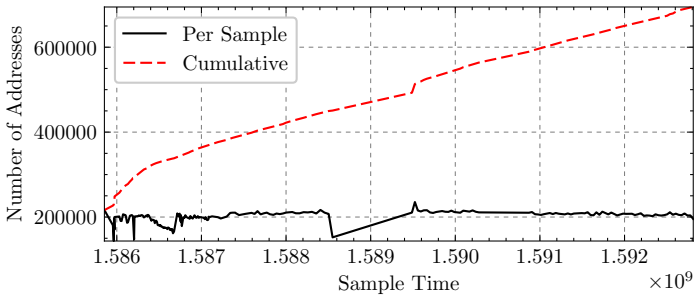


Figure 6.4: Longitudinal analysis of *unreachable* addresses collected from the network. The black line shows the unique IP addresses collected in each experiment and the red line shows the cumulative number of unique IP addresses collected in 60 days. The gap between the two lines shows that in each experiment, new IP addresses appeared in the network. Overall, we collected ≈694K unique IP addresses of *unreachable* nodes.

through their firewall. It is possible that a node is *unreachable* and running Bitcoin while it has disabled all incoming requests. In that case, our heuristic will not be able to detect the *responsive* node. Therefore, the number of *responsive* nodes that we have detected provide a lower bound estimate of *unreachable* nodes that run Bitcoin.

6.3 Analysis and Results

In this section, we present our analysis and main findings. For each of the four factors outlined in §6, we evaluate their impact on block propagation and network synchronization.

6.3.1 Unreachable Nodes

In 60 days, we collected 694,696 unique *unreachable* IP addresses, with ≈195K addresses in each experiment. Among those addresses, 615,083 (88.54%) used the default 8333 port while 79,613 (11.46%) used 9,414 unique ports. Fig. 6.4 presents the *unreachable* addresses obtained in each experiment and the cumulative number of unique *unreachable* addresses collected in all experiments. Among the total of 694,696 addresses, 163,496 (23.54%) were *responsive*, and ≈54K

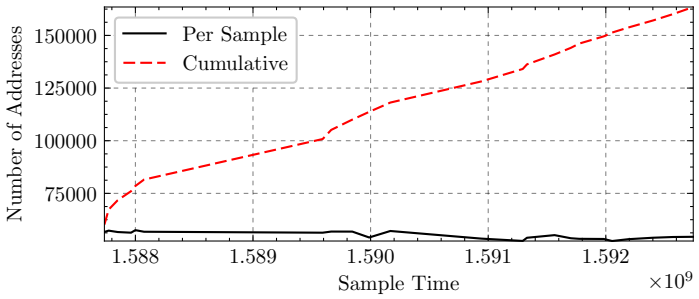


Figure 6.5: Unique IP addresses of *responsive* nodes collected in each experiment as well as cumulative. The cumulative number of *responsive* addresses follow the same trend as the *unreachable* addresses. Initially, due to an error, the experiment on *responsive* nodes was delayed by two weeks. On average, we collected ≈54K *responsive* addresses in each experiment.

(27.69%) addresses were *responsive* in each experiment. That is, ≈54K *unreachable* nodes were running Bitcoin at any time. Fig. 6.5 plots the number of *responsive* nodes.

Impact of Unreachable Nodes. Our results show that the *unreachable* network is ≈24 times larger than the *reachable* network. Considering the prevalence of *unreachable* nodes and the size gap, we analyzed the number of *reachable* and *unreachable* addresses in all ADDR messages to study the impact of the *unreachable* network on the Bitcoin network.

Since the *unreachable* addresses in the ADDR message provide no clear benefit to the network. The only useful information in the ADDR message, however, is the number of *reachable* addresses, which improve the outdegree. Therefore, propagating *unreachable* addresses may contribute to increasing the initiated (outgoing) connections failure rate.

Our results reveal that an ADDR message contains 14.9% *reachable* addresses and 85.1% *unreachable* addresses on average. That is, 85.1% of the addresses exchanged in the network provide no benefit in terms of improving the network connectivity. In the next section, we experimentally show how the *unreachable* IP addresses affect the network outdegree. Since the Bitcoin network overlay topology is anonymous, we will rely on various heuristics to aid our analysis.

6.3.2 Addressing Protocol

Per our previous analysis, we found that the *unreachable* network size is ≈24 times the *reachable* network size, and that the ADDR messages are dominated by the *unreachable* addresses. In the following, we analyze this information along with the addressing protocol to explore the impact of those nodes on the network connectivity and synchronization.

In theory, and given that there are 10K *reachable* nodes where each node has 8 stable outgoing connections, a block is received by all *reachable* nodes in five rounds ($8^5 > 10K$). In contrast, if the number of the outgoing connections drops to 2, the block can take up to 14 rounds ($2^{14} > 10K$) to propagate. Therefore, the stability of the outgoing connections significantly affects the block propagation. To analyze the stability of the outgoing connections, how this stability impacts the effective outgoing degree, and eventually understand the number of rounds it will take to propagate a block, we deployed a Bitcoin node with the recent Bitcoin Core version (v0.20.1) and analyzed variations in the number of outgoing connections.

In practice, when a Bitcoin node starts, with IP addresses populated in the *new* and *tried* tables, the node selects IP addresses from both tables with equal probability and establishes outgoing connections. If a connection drops at any time, the process of selecting an IP address from the *new* and *tried* tables is repeated until all of the outgoing slots are complete (total of eight connections). This process naturally raises two questions. (1) *How often do outgoing connections drop?* (2) *How many outgoing connection attempts are successful?*

How often do outgoing connections drop? To answer this question, we conducted an experiment using the aforementioned Bitcoin node. Upon running the node for 260 seconds, we logged the number of the outgoing connections using the Bitcoin RPC API, and reported the results in Fig. 6.6.

As shown in Fig. 6.6, the numbers of the outgoing connections are highly unstable, varying between 2 and 10 connections at any time. Aside from the eight outgoing connections specified in the Bitcoin protocol, two *feeler* connections that are not used for block or transaction exchange [106] are also observed, bringing the total sometimes to 10 at times. Overall, we observed that there was

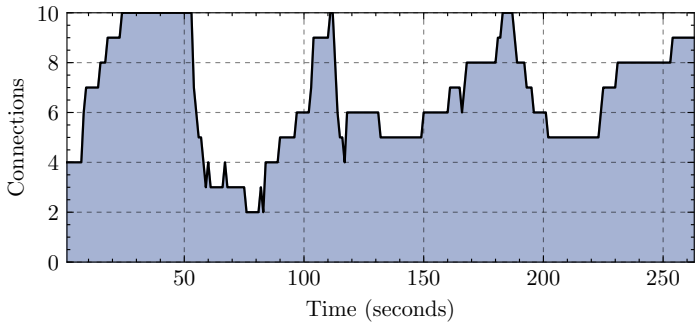


Figure 6.6: Results from the experiment conducted to analyze the stability of outgoing connections. The experiment was conducted for 260 seconds and we observed that the outgoing connections are highly unstable. The number of connections varied between 2–10 connections at any time.

less than 8 connections for ≈60% of the time, while the average was 6.67 connections.

Note that the outgoing connections can be dropped for one of several reasons, including: (1) the departure of a node from the network, or (2) connection/link failure in the physical network. Since those reasons are equally-likely for all nodes in the network, we extrapolate from this observation to the network at large. We argue that the topology of the *reachable* network is constantly changing since the outgoing connections drop frequently, as shown in this experiment. This constant change can be used to reason about the high variations in network synchronization for each block Fig. 6.1.

How many outgoing connection attempts are successful? Since now we know that the outgoing connections are unstable, the next step is to analyze the failure rate of the outgoing connection attempts. Answering this question will also help us to fully understand the impact of *unreachable* addresses in the IP tables. For this purpose, we conducted five experiments in which we started the Bitcoin node and counted the total number of the outgoing connection attempts and the number of the successful connections. Each experiment was conducted for five minutes, where we restarted our node upon each experiment to ensure settings independence.

The results of the five experiments are shown in Fig. 6.7, where we observe a high gap between the total number of the outgoing connections and the number of successful connections. On average, only 11.2% of the connection attempts were successful. Moreover, in one experiment, only 8 out

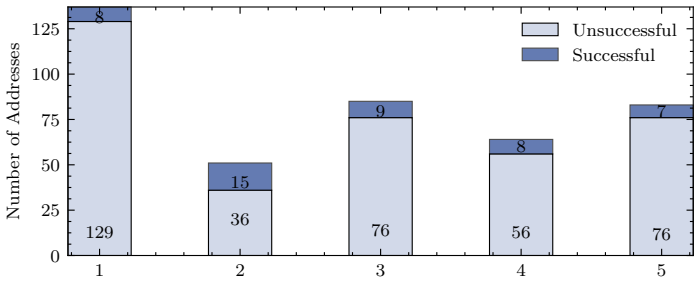


Figure 6.7: Results from the experiment conducted to analyze the success rate of outgoing connections. On average, only 11.2% attempts result in successful outgoing connections. For the second experiment, the number of successful connections appear to be 15. This is due the fact some connections were dropped after which the node made new successful connections.

of 137 attempts (5.8%) were successful. Furthermore, Fig. 6.7 shows a high diversity in the total number of the outgoing connections in each experiment. In two experiments, the total number of the outgoing slots were not filled during the experiment duration. In the second experiment (Fig. 6.7) the total number of successful connections was 15. Since the maximum outbound connections are only 8 (except the *feeler* connections), this shows that some successful connections dropped and the node tried new connections from the IP tables.

Our results show a high outgoing connection failure rate. The failure rate of 88.8% comes as a surprise, shedding light on weaknesses in the addressing protocol. Ideally, each outgoing connection should be successful. However, the low success rate in the outgoing connections shows that the network condition is far from ideal. This is in part due to the *unreachable* addresses dominating the nodes' IP tables. Moreover, since the outgoing connections are unstable, our results also show that even the *reachable* nodes may leave the network at any time due to churn. Therefore, the outgoing connections to the *reachable* nodes that have left the network also contribute to the low success rate of outgoing connections. From Fig. 6.6 and Fig. 6.7, we conclude that the *unreachable* network adversely affects the *reachable* network topology since the average node outdegree is less than the default outdegree. Extrapolating this behavior to other nodes in the network means that the average network outdegree is below the expected outdegree, and block propagation in the network

takes longer than expected, thus affecting network synchronization. Our experiments also reveal that this undesirable network state is caused by the weakness in the Bitcoin addressing protocol that allows the transmission of *unreachable* IP addresses in the ADDR message.

6.3.3 Information Relaying Protocol

As discussed in §6.1, the theoretical models of Bitcoin [76, 44, 80] assume that a Bitcoin node concurrently releases every new block to all of its connections. As such, if a node has all 125 connection slots filled and the node produces a block, the block must be relayed to all 125 connections simultaneously. In contrast, if there is delay in relaying the block to each connection, the nodes that receive the block earlier will synchronize faster. In this section, we analyze the practical implications of block relaying protocol in Bitcoin and how those aspects affect synchronization.

To study the implementation of the block relaying protocol, we inspected the *net.cpp* file in the Bitcoin Core source code [18]. We found that when the Bitcoin Core starts, it creates two threads to handle the protocol messages. On the one hand, the `SocketHandler` thread reads the incoming messages from a connected peer and stores them in the `vProcessMsg` queue. It then sends the outgoing messages to the peer from the `vSendMessage` queue using a `Write` buffer. The `ThreadMessageHandler` thread reads the messages from `vProcessMsg` and processes them using the `ProcessMessage` function.

We use the reconstructed protocol information from our analysis to draw the Bitcoin Core message handling procedure in Fig. 6.8. We further illustrate this workflow with an example. Assume two connected Bitcoin nodes, A and B, where A sends a GETADDR request to B. At B: (1) the `SocketHandler` thread will queue the GETADDR message in the `vProcessMsg` queue, (2) the `ThreadMessageHandler` thread will read the message from the`vProcessMsg` and generate the ADDR response using the `ProcessMessage` function, (3) the `ProcessMessage` function will send the ADDR response to `vSendMessage`, and (4) the `SocketHandler` thread will write the response to the socket connected with A. However, if during this process, B

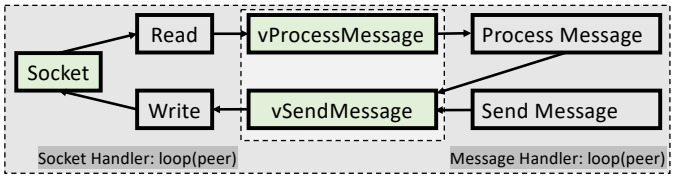


Figure 6.8: Message handling workflow. The `SocketHandler` thread loops over each peer and reads incoming messages into the `vProcessMsg` queue. It also sends outgoing messages from `vSendMessage` queue. The message handler thread reads messages from `vProcessMsg` queue and sends the output to `vSendMessage` queue.

Algorithm 3: Processing P2P Messages

Input: $\mathbb{C}_{y,z}$ where y and z are the number of connections and messages sent by each connection
foreach $y \in \mathbb{C}_y$ **do**
  foreach $z \in \mathbb{C}_{y,z}$ **do**
    if no new block **then**
      invoke `ThreadMessageHandler()` and `ThreadMessageHandler()`
      process message $\mathbb{C}_{y,z}$ (see Fig. 6.8)
    if new block **then**
      append block to `vSendMessage` queue
      increment z
      process message $\mathbb{C}_{y,z}$ (see Fig. 6.8)
return empty queue $\mathbb{C}_{y,0}$

also generates a new block and wants to send it to A, the `SendMessages` function will queue the block behind the ADDR message in `vSendMessage`, and the `ThreadMessageHandler` thread will send it to A after sending the ADDR message.

Our analysis also revealed that a Bitcoin node schedules the connections in a round-robin manner, processing one message per socket for each connection. For instance, if node B is connected to five other nodes (A–F) and wants to send a block, B will loop over each connection to send that block. Therefore, the block relaying does not follow a broadcast model assumed in prior works.

Moreover, if B wants to send a block to A, and A has already sent 3 GETADDR requests, then B will process one request per loop per connection. As a result, A will get the block after B processes 15 requests for all connections including A's three GETADDR requests. In algorithm 3, we show the pseudo-code of the information relaying in Bitcoin Core.

Empirical Evaluation. A natural effect of the round-robin relaying is that some connections receive blocks earlier than others. Therefore, there is a gap in the time a node receives a block from the network, and the time the node relays that block to all its connections. In Bitcoin, round-robin relaying is not limited to blocks only, but also applies to transactions. We note that transaction relaying also plays a critical role in network synchronization, due to the deployment of the "compact block" relay method in Bitcoin [30].

In the compact block relay method, a node only receives the block header and transaction identifiers from the sending node. The node then reconstructs the block using transactions from its memory pool [30]. If some transactions are missing from the memory pool, the node requests those transactions from the sending node to fully reconstruct the block. If those transactions are delayed, the node cannot reconstruct the block and remains behind the blockchain. Therefore, transaction relaying also plays a significant role in network synchronization. Taking this into account, we measure the delay incorporated by round-robin transaction and block relaying.

In order to empirically measure the delay in transaction and block relaying, we set up a *reachable* Bitcoin node with 8 outgoing connections and 17 incoming connections. We then measured (1) the time at which the node received a transaction or a block from the network, and (2) the time at which that transaction or block was relayed to the last connection. We call the difference between the two events as the relaying time. Naturally, a high relaying time is undesirable for network synchronization since the receiving node stays behind the blockchain during that period. We collected the relaying time for transactions and blocks from the *debug.log* file in the Bitcoin Core data directory. The log file captures each event at one second interval. If a transaction or a block is received and relayed within a second, the timestamp for the two events is the same.

Fig. 6.9 and Fig. 6.10 report the results collected by our node over two days. Fig. 6.9 shows that the average relaying time for all blocks was 1.39 seconds, with a minimum and a maximum relaying time of 0 and 17 seconds, respectively. Fig. 6.10 shows that the average relaying time for all transactions was 0.45 seconds, with a minimum and a maximum relaying time of 0 and 8

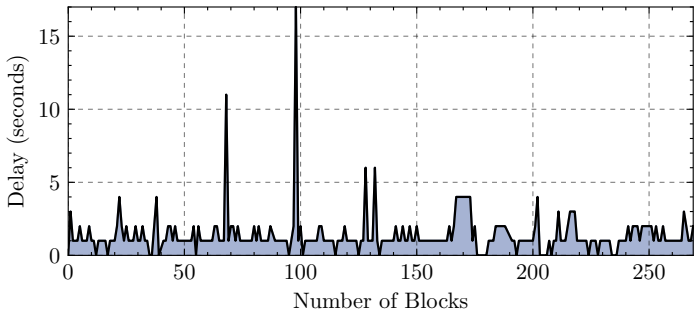


Figure 6.9: Delay between the time of receiving block and the time at which the block is relayed to the last connection. On average, it takes 1.39 seconds to relay blocks to all connections

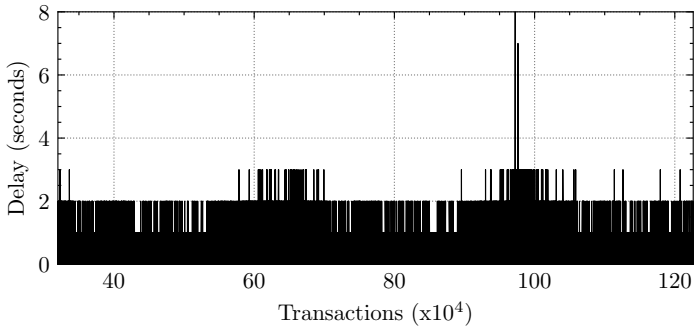


Figure 6.10: Delay between the time of receiving the transaction and the time at which the transaction is relayed to the last connection. The average delay is ≈0.45 seconds.

seconds, respectively. From these results, we conclude that the round-robin relaying adds delay in relaying transactions and blocks to the nodes, thus affecting network synchronization. We further observed that during the relaying process, the sending node does not prioritize the *reachable* nodes over the *unreachable* nodes; a distinction that can be easily made by observing the incoming and outgoing connections. Since *reachable* nodes enable network synchronization by relaying blocks to other *reachable* nodes. As such, if the *reachable* nodes are among the last connections to receive transactions or blocks, the network synchronization is affected considerably; e.g., delay of up to 17 seconds in some cases, as shown in Fig. 6.9.

6.3.4 Network Churn

Since the Bitcoin network is *permissionless*, nodes can leave the network at any time, causing a churn. If *reachable* nodes leave the network, the average network outdegree decreases which affects synchronization. In this section, we measure the impact of churn on network synchronization.

If a node leaves the network, at least 8 outgoing connections drop in the *reachable* network [57, 74]. Although, intuitively, it might appear that if the number of *reachable* nodes in the network is constant, the average network outdegree must be constant as well. However, this assumption can be false once we take churn into account.

To illustrate this problem with an example, consider a *reachable* node A with 8 stable outgoing connections. Next, assume that the node A leaves the network at time t_x, and all its outgoing connections drop. Further assume that at the same time t_x, another *reachable* node B joins the network. As shown in Fig. 6.7, a node takes time to successfully establish 8 outgoing connections. If we assume that node B successfully makes 8 outgoing connections by the time t_y, then during $\Delta = t_y - t_x$, the network will have a fewer number of outgoing connections despite the same network size. Accordingly, the network will have a smaller outdegree during $\Delta = t_y - t_x$.

Moreover, even after establishing 8 outgoing connections, node B will take time to catch up with the blockchain by requesting blocks that are not present in its local blockchain. Once node B synchronizes with an up-to-date blockchain, only then it can help in relaying the latest blocks to other *reachable* nodes. In other words, during the time where node B is not up-to-date, it does not contribute to the network synchronization. Moreover, the process of catching up with the blockchain can take a few days if node B joins the network for the first time.

In an experiment to evaluate a node's capability for contributing to the network synchronization, we set up a Bitcoin node with an up-to-date blockchain. In the *debug.log* file, we observed that our node was relaying the latest blocks to its connections, thus helping other nodes to synchronize. Next, we restarted the node and recorded the time the node took to synchronize again. Our results show that the node took 11 minutes and 14 seconds to synchronize with the network and regain

Algorithm 4: Creating Binary Matrix for Churn

```
Input: Reachable addresses U and sampling time T
Initialize: Binary matrix M
foreach ip i ∈ N do
    foreach sample j ∈ T do
        if ip in sample then
            M_{i,j} ← 1
        if ip not in sample then
            M_{i,j} ← 0
return Binary matrix M
```

the ability of relaying blocks to its connections. Most of this time was spent on establishing stable outgoing connections and synchronizing on the latest block. Alternatively, if the node stayed offline for several days, it would have taken a longer time to synchronize.

From our results, we concluded that the departure of synchronized nodes and arrival of new nodes are unfavorable for network synchronization. With the departure of existing nodes and the arrival of new nodes, a high churn would lead to poor network synchronization. In the following, we model Bitcoin churn and measure the nodes arrival and departure rates.

Modeling Network Churn. To model churn in the *reachable* network, we systematically evaluate (1) the arrival time of a new *reachable* node, (2) the departure time of that node, and (3) whether a node rejoins the network after departure.

For this analysis, we sampled all of the *reachable* addresses (N_r in algorithm 1) as an object T, with the network sampling time as the object keys and the *reachable* IP addresses corresponding to the sampling time as values. Next, we collected all 28,781 unique *reachable* IP addresses in a list U, and checked the presence of each IP address for the sorted keys in T. We then initialized a zero matrix $\mathbf{M}$ in which each row denoted an IP address in U, and each column denoted the sorted sampling time from T. If an address was found at the sampling time, its index in $\mathbf{M}$ is changed from 0 to 1. In algorithm 4, we outline the procedure of obtaining the binary matrix $\mathbf{M}$, and in Fig. 6.11, we plot its binary image to provide a high level overview of churn in the *reachable* network. The colored region shows the presence of an IP address in the network.

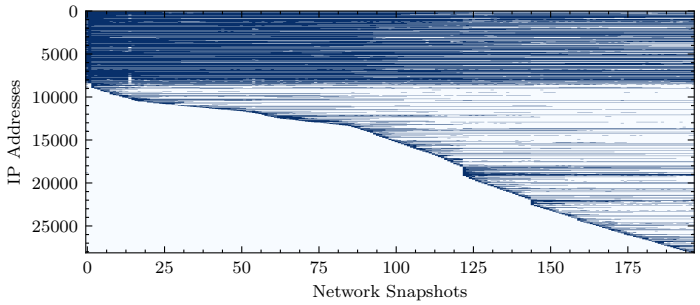


Figure 6.11: Binary matrix plot. Value 1 is marked 1 while value 0 is marked white. For a given IP address, end-to-end horizontal line shows that the address was connected in each experiment.

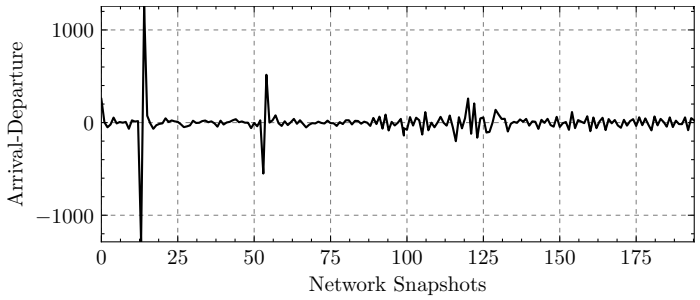


Figure 6.12: The difference between the number of nodes that leave the network and the new nodes that join. Overall, the arrival rate and the departure rate of nodes is roughly constant.

From Fig. 6.11, we observe the following. (1) A dominant white region in the bottom left shows that a significant number of new nodes join the network. (2) A majority of lines does not cover the entire x-axis after the starting point, indicating that a significant number of nodes leave the network. (3) The reappearance of a few lines on the x-axis shows that some nodes rejoin the network after leaving. (4) A few lines covering the entire x-axis show that a few nodes are always present in the network. More precisely, we found 3,034 nodes that were always present in the network.

We empirically analyze nodes that join and leave the network daily. For that purpose, we take two consecutive network snapshots and count the addresses that change in them by comparing each column in $\mathbf{M}$ (algorithm 4) with the previous column. A change in the row value from 1 to 0 indicates a node departure, while a change from 0 to 1 indicates a node arrival. In Fig. 6.12, we report

our results showing that the difference between the arrival of new nodes and departure of existing ones is small. We also note that ≈708 nodes (8.6% *reachable* nodes) leave the network everyday, replaced by an equal number of new nodes. As explained earlier, replacing departing nodes with new nodes affects synchronization, since the new nodes take time to receive the blockchain before contributing to synchronization. Therefore, the arrival rate of 8.6% shows that a significant number of non-synchronized nodes appear in the Bitcoin network each day.

Measuring Departure of Synchronized Nodes. Among the *reachable* nodes that leave the network, not all nodes are synchronized with an up-to-date blockchain. Moreover, if a non-synchronized node leaves the network, the average network synchronization would increase. Therefore, a logical question would be to determine *how many synchronized nodes leave the network?* By contrasting the departure of synchronized nodes between 2019 and 2020, we can confidently answer the problem of decreasing synchronization shown in Fig. 6.1.

In order to experimentally evaluate the departure rate of the synchronized nodes, we used the Bitnodes dataset which we are collecting since September 2019, at 10 minutes interval. To contrast our results with Fig. 6.1, we divided our dataset into two segments, consistent with the segmentation used for Fig. 6.1 . The first segment included data from September to December 2019, and the second segment included data from January to April 2020. For each segment, among the total number nodes that leave the network in 10 minutes, we counted the number of synchronized nodes.

Our results show that the average number of synchronized nodes that left the network in 10 minutes was 3.9 (≈4) in 2019, which then increased to 7.6 (≈8) in 2020. In other words, the departure of synchronized nodes nearly doubled in 2020, thus decreasing the total number of nodes that contribute to network synchronization. Since the Bitcoin addressing protocol and the block relaying protocol did not change between 2019 and 2020, it is safe to assume that their impact on network synchronization has remained constant. As a result, we conclude that the most significant change in the network is the churn among the synchronized nodes, which has doubled in 2020, and its impact on synchronization is clear in Fig. 6.1.

Key Takeaways. To put our analysis in context, we observe an increasing churn among the synchronized nodes, especially in 2020. When the synchronized nodes leave the network, the outgoing connections of their peers drop below the default threshold, upon which those nodes try new connections from their IP tables. Due to weaknesses in the addressing protocol, the IP tables of *reachable* nodes are dominated by *unreachable* addresses, causing high failure rate of the outgoing connections (Fig. 6.7). Due to the departure of synchronized nodes and delayed connection recovery of their peers, we observe two consequences. First, fewer nodes are left in the network that contribute to network synchronization. Second, during the failed connection attempts time, the average network outdegree remains low, further slowing the block propagation.

We observe 73 malicious nodes that propagate *unreachable* addresses in the network to overwhelm the IP tables of other *reachable* nodes and increase the connection failure rate. Moreover, the information relaying protocol does not prioritize block propagation to the *reachable* nodes over *unreachable* nodes, which led to block relaying delay by up to 17 seconds in some cases (Fig. 6.9). In summary, all these factors play a role in weak network synchronization in the Bitcoin network.

6.4 Improving Bitcoin Network Synchronization

Based on our measurements and analysis, in this section, we propose refinements to the Bitcoin Core design in order to improve the network synchronization.

Refining the Addressing Protocol. Since the Bitcoin network is *permissionless*, therefore, we cannot prevent churn, despite its implications on network synchronization. However, we can help the network to recover from the departure of synchronized nodes by tailoring the addressing protocol. Since relaying *unreachable* addresses in the ADDR message does not suit the network, therefore, the source code can be modified to only select IP addresses from `tried` table for the ADDR message. This will significantly improve the success rate of outgoing connections.

Refining the `tried` Table. We note that selecting IP addresses from the `tried` table only partly solves the problem. The churn analysis shows that 708 *reachable* nodes leave the network

everyday. Despite their departure, their IP addresses stay in the `tried` of their connections. A *reachable* IP address is removed from the `tried` table if: (1) more than 10 connection attempts to that address fail in one week, or (2) the IP address is in the table for 30 days [55]. As such, even if we apply the policy of sending IP addresses from the `tried` table, if the sending node has not evicted the IP address from `tried` table, it will not be useful for the receiving node.

To make the IP address eviction policy more efficient, we revisit the threshold of retaining an IP address in the `tried` table for 30 days. In the Bitcoin Core source code, we did not find any justification for retaining an IP address for 30 days. We believe the underlying assumption of the early developers was that the average network life time for a majority of nodes will be 30 days. To revise this policy, we use results from Fig. 6.11 to provide a more realistic estimate of a node's lifetime. Our results show that the average network lifetime of a node is only 16.6 days. Therefore, by reducing the limit of 30 days to 17 days, we can increase the eviction rate of IP addresses that leave the network. With an increased eviction rate, we can increase the percentage of *reachable* addresses in the ADDR message, thus increasing the success rate of outgoing connections.

Prioritizing Block Relay. Network synchronization can also be improved by prioritizing block relaying to the *reachable* nodes. In the current implementation, a sending node does not distinguish between incoming connections and the outgoing connections. While incoming connections can be from both *reachable* and *unreachable* nodes, the outgoing connections are always established with the *reachable* nodes. As such, if a new block is mined or received from any connection, the node should first relay that block to all the outgoing connections, thereby increasing block propagation among the *reachable* nodes. Moreover, if there is a queue of requests (*i.e.*GETADDR messages) in the `vSendMessage` queue, then the block can be prioritized over those requests to minimize unusual delay in block relaying (*i.e.* 17 seconds in Fig. 6.9).

By incorporating these changes, the Bitcoin network synchronization can be significantly improved despite the high network churn. If a synchronized node leaves the network, all its peers will immediately recover their outgoing connections, thus maintaining the average network outdegree. If the

synchronized node does not rejoin the network, its IP addresses will be removed from the `tried` tables in 17 days. This will prevent the undesirable relaying of the address in the ADDR message. Finally, by prioritizing block relaying to the outgoing connections, we can ensure that the *reachable* network synchronizes quickly over a newly published block. By maintaining the network outdegree and ensuring faster block relay, the undesirable consequences of churn on Bitcoin network synchronization can be significantly minimized.

6.5 Summary

In this work, we conduct the root cause analysis of deteriorating Bitcoin network synchronization. Through measurements and analysis, we show that the deteriorating network synchronization is due to (1) a large number of *unreachable* nodes, (2) weaknesses in the network addressing protocol, (3) the delay incorporated by round-robin block relaying, and (4) a high churn among the *reachable* nodes. Among all these factors, we note that the most significant factor in the recent months is the churn among the synchronized *reachable* nodes.

Our work also exposes two important characteristics of the real world Bitcoin network that have not been concretely evaluated in prior works. First, we note that the Bitcoin network is asynchronous which can be exploited to mount new attack strategies to violate the blockchain consistency. Second, the impact of churn on network synchronization has not been formally characterized in the Bitcoin security model. In the following two chapters, we elaborate on these findings and highlight the security risks associated with the asynchronous network and the network churn.

CHAPTER 7: *HASHSPLIT*: EXPLOITING ASYNCHRONY TO VIOLATE BLOCKCHAIN CONSISTENCY AND CHAIN QUALITY

Our attack surface analysis reveals that the application-specific design choices and the network layer inconsistencies can be exploited to attack blockchain systems. So far, we have analyzed these two layers independently by showing attacks that are specific to the blockchain application constructs or the P2P network. In this work, we will consolidate our prior insights and present a novel attack that results from a combination of the application-specific policies and the network layer inconsistencies. We note that in Bitcoin, if a miner receives two valid blocks linked to the same parent block (*i.e.* a fork), the miner extends the block that is received earlier [76]. If this mining policy is applied in an asynchronous network, the probability of forks increases.

If all miners behave honestly, such forks can be easily resolved. However, a malicious miner can exploit the mining policy and the asynchronous network to launch an attack that prevents the fork resolution. In this work, we investigate the feasibility of such an attack in the Bitcoin network, and the attack implications on the fundamental blockchain properties. However, this analysis has several challenges including (1) theoretical modelling of Bitcoin network that specifies the blockchain *consistency*, (2) large-scale measurements to identify a subset of mining nodes among all the network nodes, (3) experimentally validating asynchronous communication among the mining nodes, and (4) curating attack strategies that favor the adversary.

7.1 Contributions

We overcome these challenges and present the *HashSplit* attack that allows an adversary to violate the blockchain *consistency* and chain quality with a high probability.

1. We construct the Bitcoin ideal world functionality to formally specify the two notable properties of the Bitcoin ledger; the common prefix property and the chain quality property [44]

(§7.2). The ideal world functionality faithfully models the expected functionality of a correct Bitcoin implementation across prevalent deployments in real world Bitcoin network.

2. We deploy crawlers in the Bitcoin network and connect with over 36K IP addresses in five weeks (§7.3). We develop heuristics to identify the mining nodes and identify 359 IP addresses of the mining nodes using those heuristics (§7.4).

3. We measure the block propagation patterns in the Bitcoin network (§7.5) and show that that the average Bitcoin block time is 9.97 minutes during which only ≈39% nodes receive the latest block, indicating a high propagation delay and weak synchronization. Moreover, through a fine-grained analysis of the block propagation among the mining nodes we show that the Bitcoin network is asynchronous in the real world (§7.5.1).

4. We show the effect of the asynchronous execution by presenting the *HashSplit* attack which allows an adversary to violate the common prefix and chain quality properties of the Bitcoin blockchain. We also propose attack countermeasures by developing a Bitcoin Core version that closely models the ideal functionality and resists the network asynchrony [3].

7.2 The Bitcoin Ideal World Functionality

In this section, we present the Bitcoin ideal world functionality, which we later contrast with real world measurements to present the *HashSplit* attack. The Bitcoin white paper by Nakamoto assumed a network where each node possessed the capability of solving PoW (1 CPU=1 Vote) [76]. However, over time, the PoW difficulty significantly increased, allowing only a few nodes to solve it. This change occurred due to large mining pools that drive implicit forms of centralization [108]. Since there are fewer mining pools than the number of Bitcoin users, therefore, there are fewer mining nodes in the network. Realizing these changes, we formally define the Bitcoin ideal functionality to characterize the existing Bitcoin operative model, including the distinctive functionality of the mining and non-mining nodes. The formulation of our ideal functionality is inspired by models proposed in [44, 80] with necessary adjustments to incorporate the mining centrality.

Ideal World Functionality of Bitcoin

Input: Nodes $\mathbb{N}$ including miners M, blockchain 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 the round from $P_i \in M$, $\mathcal{F}$ extends the chain $C \leftarrow b_{r+1}$.
- 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 $C \leftarrow b_{r+1}$.
- If $P_i \in M$ receives b_{r+1} from another $m_j \in M$ in round, then P_i stops its own computation for b_{r+1}, extends the chain $C \leftarrow b_{r+1}$, 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 $C_1 \leftarrow b_{r+1}$ $C_2 \leftarrow b_{r+1}^{'}$. Both C_1 and C_2 have an equal length.
- If $P_i \in M$ receives multiple inputs for the same parent block (*i.e.,* $b_{r+1} \preceq b_r$ and $b_{r+1}^{'} \preceq b_r$), P_i gives time-based precedence to the blocks. *i.e.,* b_{r+1} is received at t_1 and $b_{r+1}^{'}$ is received at t_2, where $t_2 > t_1$, then P_i only accepts b_{r+1} and discards $b_{r+1}^{'}$ by treating it as an orphaned block. P_i extends the chain $C \leftarrow b_{r+1}$ 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 $C_1 \leftarrow b_{r+1}$ $C_2 \leftarrow b_{r+1}^{'}$. Both C_1 and 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 $\text{VIEW}_C^{P_i}$. $\mathcal{F}$ 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 $\text{VIEW}_C^{P_i}$. If $\mathcal{F}$ receives a single ledger 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.,* C_1 and C_2) from one or more $P_i \in \mathbb{N}$, $\mathcal{F}$ prunes k blocks from C_1 chain and verifies if $C_1^{\lceil k} \preceq C_2$ (*i.e.,* two chains share a common prefix). To evaluate $\mathcal{Q}_{cq}$, $\mathcal{F}$ selects the longest chain among C_1 and C_2, and computes the experimental value of μ_i. If $\mu_i - \mu_i^{'} = \epsilon$ (negligible), $\mathcal{F}$ assumes $\mathcal{Q}_{cq}$ is preserved. Otherwise, $\mathcal{Q}_{cq}$ is violated and some $P_i \in M$ has maliciously contributed more blocks than its hash rate.

Figure 7.1: he Bitcoin ideal world functionality, closely modeled on the practical implementation of Bitcoin as we largely see it. Only mining nodes $P_i \in M$ participate in the block race. The communication model follows the primitive specifications of [76, 44]. We also distinctly characterize the behavior of mining ($P_i \in M$) and non-mining ($P_i \notin M$) nodes when they receive blocks.

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 [44] 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. 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 (also defined as the consistency property in [80]) and the chain quality property of the Bitcoin blockchain.

Common Prefix Property. The common prefix property $\mathcal{Q}_{cp}$, with parameter k specifies that for

any pair of honest nodes P_1 and P_2, adopting the chains $\mathcal{C}_1$ and $\mathcal{C}_2$ at rounds $r_1 \leq r_2$, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$. In this context, an honest node is a node that respects the ideal functionality. $\mathcal{C}_1^{\lceil k}$ denotes the chain obtained by pruning k blocks from $\mathcal{C}$, and $\preceq$ is the prefix relationship. For transaction confirmation, the common prefix property must hold for 6 blocks ($\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$ for $k = 6$) [24].

Chain Quality Property. The chain quality property $\mathcal{Q}_{cq}$ with parameters μ and l specifies that for any honest node P_i with chain $\mathcal{C}$, it holds that for any l consecutive blocks of $\mathcal{C}$, the ratio of honest blocks is at least μ. $\mathcal{Q}_{cq}$ ensures that the contribution of P_i in $\mathcal{C}$ is proportional to h_i. Moreover, $\mathcal{Q}_{cq}$ assumes that no $P_i \in M$ acquires more than 50% hash rate [39, 67, 49, 53, 99].

Using these properties, we define the Bitcoin ideal world functionality in Fig. 7.1. Our formulation assumes $P_i \in \mathbb{N}$ as "interactive Turing machines" (ITM) that execute the Nakamoto consensus for l rounds, arbitrated by a trusted party $\mathcal{F}$. A round is a time in which each $P_i \in M$ is mining on the same block. For any $P_i \in M$, a round terminates when the $\text{VIEW}_{\mathcal{C}}^{P_i}$ is updated with a new block. The network $\mathbb{N} \times \mathbb{N}$ is a fully connected such that when a block is released by any $P_i \in M$ at t_1, all nodes receive it at the next time step t_2. As a result, the network exhibit a *lock-step* synchronous execution [83]. Due to varying roles in the system, the mining nodes $P_i \in M$ and the non-mining nodes $P_i \notin M$ have unique operations. For instance, when a $P_i \in M$ receives two valid blocks for the same parent block, it gives time-based precedence to the block received earlier. The block received later is discarded. However, when a $P_i \notin M$ receives two valid blocks, it creates two concurrent branches of the chain and waits for the next block to extend one of them. The ideal world functionality in Fig. 7.1 is consistent with the application rules in Bitcoin Core.

Ideal Functionality Proof. In the following, we provide the proof for the ideal world functionality.

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

Proof. Prior to the proof sketch, we present some practical considerations for the execution model.

In Bitcoin, the average duration of a round is 10 minutes (600 seconds) and the parameter k for the common prefix is 6 blocks [24]. 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 6 blocks. For this condition to hold, 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 topology, the remaining miners receive the two blocks at t_1. As shown in Fig. 7.1, 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:

$$P(x|\lambda) = \left(\frac{e^{-\lambda}\lambda^x}{x!}\right)^k \tag{7.1}$$

Now assume a random variable X which represents an event that $\mathcal{C}_1^{\lceil 6} \npreceq \mathcal{C}_2$ 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 (7.1)) becomes:

$$P(X) = \left(\frac{e^{-\lambda}\lambda^x}{x^2(x-1)!}\right)^k \tag{7.2}$$

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

To ensure the chain quality property, $\mathcal{F}$ specifies that no h_i for $P_i \in M$ has more than 50% hash rate. Otherwise, $\frac{h_i}{H}$ does not hold and $\mathcal{F}$ aborts. Moreover, in the winning chain, the number of blocks contributed by the honest miners is proportional to their hash rate. For instance, in a chain length of $l = 6$ rounds in which 6 consecutive blocks are produced, a miner with 14.3% hash rate should be able to contribute 1 block (μ_i). If a miner faithfully respects the protocol in Fig. 7.1, its probability of contributing 1 block becomes $k\frac{h_i}{H}$. Plugging in the experimental values,

the probability is 0.999 ($\mu_i^{'}$. Therefore, $\mu_i - \mu_i^{'}$ is 0.001. This is a negligible probability (ϵ) which we defined in the ideal world functionality (Fig. 7.1). □

Key Takeaways. The ideal world functionality, in Fig. 7.1, characterizes the *modus operandi* of Bitcoin. Compared to the prior theoretical models [44, 80], we distinctly define the mining nodes and non-mining nodes and characterize their unique roles in the system. In the rest of the chapter, we perform a data-driven study to investigate (1) the size $|M|$ of the mining nodes, (2) the synchronization patterns in the network to understand how closely Bitcoin follows the ideal functionality, and (3) show deviations to construct the *HashSplit* attack.

7.3 Data Collection

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

7.3.1 Bitcoin Peer-to-Peer Network

There are two types of Bitcoin full nodes, namely the *reachable* nodes and the *unreachable* nodes. The *reachable* nodes establish outgoing connections as well as accept incoming connections from other *reachable* and *unreachable* nodes. The *unreachable* nodes (often behind a NAT [35]) only establish outgoing connections. For simplicity, we can characterize the Bitcoin network between the *reachable* space and the *unreachable* space, as shown in Fig. 7.2.

It is argued that mining pools prefer to host their mining nodes in the *unreachable* space due to security concerns [35]. As such, if we assume that *all* mining nodes exist in the *unreachable* space, it implies that mining nodes cannot accept incoming connections with 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

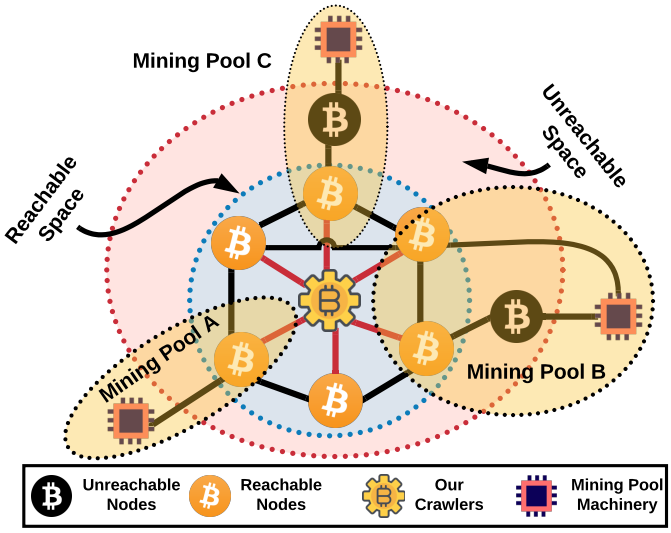


Figure 7.2: An illustration of our data collection system contextualized in the Bitcoin framework. Mining pools can have reachable (Mining Pool A), unreachable (Mining Pool C), or both (Mining Pool B) types of nodes. Note that unreachable nodes cannot connect with each other. Therefore, a block must appear in the reachable space to reach other miners. Our crawlers directly connect with all the reachable nodes to receive their blocks directly.

the ideal world functionality. 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 [34]. To further understand these arguments, we reached out to the Bitcoin Core developers and authors of prior work above. We learned that there is no empirical evidence to support the argument that all the mining nodes exist in the *unreachable* space. In fact, mining pools host both *reachable* and *unreachable* mining nodes. From those discussions, we made the following characterizations.

(1) Mining pools typically host both *reachable* and *unreachable* nodes. (2) Since two *unreachable* nodes cannot directly connect to each other, blocks between the *unreachable* nodes are relayed by the *reachable* nodes. (3) *Reachable* nodes are responsible for relaying blocks and maintaining the network synchronization. (4) This block relaying method is followed even when miners use fast relay networks such as FIBER or Falcon [75]. (5) Since the *reachable* nodes are the entry points for a block in the *reachable* network (Fig. 7.2), we can mark those entry points and treat them as

the mining nodes by connecting to all the *reachable* nodes.[1] (6) The frequency of relaying blocks can be used to estimate the hashing power of the mining pool behind a *reachable* node.

Using these insights, we set up a data collection system to connect with all the *reachable* nodes. Based on the prior work, we noticed that the number of *reachable* nodes in Bitcoin can vary between ≈7K to ≈9K addresses at any time. However, unlike [92], we did not want to rely solely on Bitnodes [23] for data collection since Bitnodes does not disclose the mining nodes. Instead, we developed our own data collection system and customized it to our desired specifications.

Data Collection System. We deployed eight crawlers in the Bitcoin network to connect with all the *reachable* nodes. At each crawler, we mounted a NodeJS implementation of the RPC client-server module for data collection and analysis. We also set up a *node manager* and installed the *peers.dat* parser on it. The *node manager* 1) connected to all the crawlers, 2) provided them the list of IP addresses to connect with, 3) obtained the JSON data from each crawler, 4) applied techniques to identify the mining nodes, and 5) measured the block propagation patterns at specified intervals to monitor the network synchronization. In five weeks, we connected to 36,360 unique IP addresses, including 29,477 IPv4, 6,391 IPv6, and 522 Tor addresses. Fig. 7.2 provides an illustration of our data collection system in the context of Bitcoin peer-to-peer network. We connected to all the *reachable* nodes in the Bitcoin network and whenever a mining pool released a block in the *reachable* space, our crawlers marked that node and analyzed the block propagation.

At each crawler, we used high-speed fiber-optic Internet with a 1GBps connection to minimize propagation delay that could affect measurement results. After five weeks, we discontinued the experiment since we did not observe any increase in the number of mining nodes. Continuing the experiment would not have yielded more meaningful results while continuously surmounting the connectivity cost. At any moment, the crawlers were connected to ≈10K IP addresses, consuming significant bandwidth. Constrained by resources, we limited the experiment duration until we had

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

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

Figure 7.3: A sample JSON output when a block is received by our crawler from a peer. Here, "addr" is the IP address of the peer to which the crawler is connected to, "synced_headers" is the height of blockchain header at which the crawler has synchronized with the node, and "inflight" is the block that the node is transmitting to the crawler.

sufficient data to motivate for the *HashSplit* attack.

Ethical Considerations. Complying with the ethical practices, we did not remove all crawlers immediately. Instead, we slowly terminated connections at each crawler to avoid risking any significant effect on the network's health. To avoid influencing block propagation and network synchronization, we disabled block forwarding at our crawlers. Finally, we want to emphasize that our crawlers were merely *listeners* in the network since they only logged the information that was willingly disclosed by their connections. We did not send any measurement probes other than what is acceptable within the Bitcoin network (*i.e.* GETDATA message in response to a block INV message). When a mining node sent a block through INV or CMPCTBLOCK method, our crawlers logged the information and took the network snapshot at the time of receiving the block.

7.4 Identifying the Mining Nodes

Prior works used INV messages to detect mining nodes [5], which is storage intensive and less accurate since the deployment of CMPCTBLOCK method [30]. To overcome this limitation, we used the RPC API to sample the network and developed **Heuristic 1** to detect the mining nodes.

The Bitcoin RPC API command *getblockchaininfo* provides information about the latest block on the blockchain tip. We deployed a socket listener at the RPC client-side implementation to record the arrival of a new block from a mining node. When a new block was received, it generated an interrupt on the listener which invoked the *getpeerinfo* API. The *getpeerinfo* renders the up-to-date interactions with all connected peers. A sample interaction with one peer is shown in Fig. 7.3 and the key variables to note are "addr", "lastsrecv", "synced_headers", and "inflight." "addr" is the connected peer's IP address, "lastrecv" is the latest UNIX timestamp at which the peer relayed any information, "synced_headers" is the peer's blockchain height, and the "inflight" is the block relayed by the peer. Viewed through the lens of our ideal world functionality (Fig. 7.1), "synced_headers" 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" and "inflight". Leveraging this useful information, we developed **Heuristic 1** to detect the mining nodes in the Bitcoin network.

Heuristic 1. 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 denotes the IP address of the mining node $P_i \in M$'.

Heuristic 1 is a mapping between the information exposed by the RPC API and the Bitcoin network traffic of a crawler. For more clarity on **Heuristic 1**, revisit Fig. 7.3 that shows a sample interaction of a crawler connected to peers in $\mathbb{N}$ with its blockchain tip $\mathcal{C}$ = 570366 ("synced_blocks" = 570366). When the crawler receives an update "570367" from *getblockchaininfo*, it checks $\text{VIEW}_{\mathcal{C}}^{P_i}$ of all its peers using *getpeerinfo*. One information sample of a peer is shown in Fig. 7.3. For each peer, the crawler checks if "synced_headers" value is 570367 ($\mathcal{C}+1$). When the mining node relays a block, the "inflight" value is also set to the block height ($\mathcal{C}+1$). The example in Fig. 7.3 shows that the "inflight" value is "570367", hence the "addr" is the mining node's IP address.

We performed two experiments called the "locality-inference experiment" and the "full-scale experiment". For each experiment, we used **Heuristic 1** to detect the mining nodes.

Locality-inference experiment. In the locality-inference, we evaluate an intuitive hypothesis

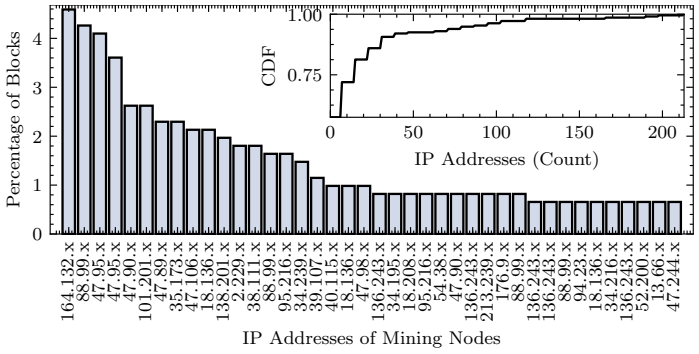


Figure 7.4: Locality-inference experiment result. The histograms show the percentage of blocks contributed by the top IP addresses. We mask thelast two octets to preserve privacy. Inside the plot, there is a CDF showing the distribution of IP addresses with respect to the total blocks produced.

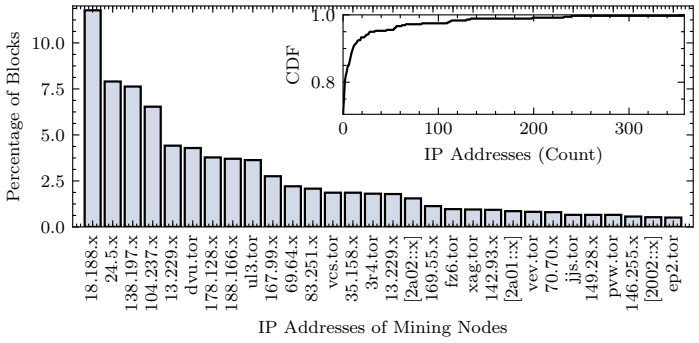


Figure 7.5: Full-scale experiment result. The histograms show the percentage of blocks contributed by top IP addresses. We mask the last two octets to preserve privacy. Inside the plot, there is a CDF showing the distribution of IP addresses with respect to the total blocks produced.

proposed in [35, 5], which assumes that mining pools prefer to host their mining nodes within the same network prefix to minimize the propagation delay. Therefore, if we obtain a list of IP addresses that use the ASIC mining hardware and find a corresponding prefix mapping in our Bitcoin network, we can narrow our measurements to fewer nodes in $\mathbb{N}$. This would reduce the network size $\mathbb{N}$ to $\mathbb{N}'$, where $|\mathbb{N}'| < |\mathbb{N}|$, and reduce the overhead of detecting the mining nodes.

For this experiment, we used the Internet scanner *Shodan* to obtain IP addresses of machines that were running Bitcoin mining hardware. *Shodan* uses hardware fingerprinting to collect IP

addresses that disclose their hardware signatures [68]. Typically, the Bitcoin miners use custom hardware called *AntMiner* for PoW hashing [22]. From *Shodan* and *Censys*, we obtained 1,714 IP addresses. Next, we checked how many Bitcoin nodes $\mathbb{N}$, were hosted within the same subnet of those IP addresses. We found a match for 1,033 IP addresses ($\mathbb{N}^{'}$ = 1,033, where $|\mathbb{N}^{'}| < |\mathbb{N}|$), indicating that 1,033 Bitcoin nodes were hosted in the same subnet of the mining hardware. We deployed a single crawler that connected to all 1,033 IP addresses and collected their data.

Full-scale experiment. The locality-inference experiment, however intuitive, has few limitations. The mining pools are multi-homed [21]. Therefore, the centrality of the mining hardware in one place may not be representative that a lot of blocks will be produced from the mining node hosted within the same prefix. Moreover, *Shodan* and *Censys* only provide IPv4 addresses and do not provide information about the mining hardware using IPv6 nor Tor. As such, the results of the locality-inference experiment may not be highly reliable.

To validate the strength of the assumption behind the locality-inference experiment, and to obtain more reliable results, we performed a full-scale experiment in which we connected to all nodes, $\mathbb{N}$, and applied **Heuristic 1** to identify M. In the full-scale experiment, we did not rely on any apriori supposition regarding the mining nodes (*i.e.* locality distribution). We measured the entire network and its behavior over time.

Results. Results of the full-scale experiment are reported in Fig. 7.5, while the results of the locality-inference are reported in Fig. 7.4. We mask the last two octets of each IP address to preserve anonymity. Key findings from the two experiments are summarized below.

1. In the locality-inference experiment, we found 214 unique mining nodes. Among them, 26 nodes produced 50%, 92 nodes produced 80%, and 153 nodes produced 90% of the total blocks. Fig. 7.4 shows the distribution of the mining nodes.

2. In the full-scale experiment, we observed 359 mining nodes. Among them, 256 (68.5)% were IPv4, 29 (8.1%) were IPv6, and 65 (23.4)% were Tor addresses. We noticed that

mining pools use Tor addresses to safeguard their mining nodes against routing attacks.

3. In the full-scale experiment, 31 nodes produced 80% and 60 nodes produced 90% blocks.

4. We found 13 IP addresses that overlapped between the two experiments, which is only 3.6% $|M|$. Due to the weak overlap, we concluded that either the Internet scanners were not able to accurately or completely determine the mining hardware, or the mining pools did not host the mining nodes within the same prefix of the mining hardware. Hence, the locality-inference is not highly useful in locating the mining nodes.

Given the current Bitcoin network size, mining nodes detection is a non-trivial task requiring a large-scale data collection and sophisticated detection techniques. However, mining nodes detection not only contributes to the *HashSplit* attack but also lowers the cost of launching spatial, temporal, and spatio-temporal partitioning attacks discussed in the prior works [5, 92].

7.5 Network Synchronization

In this section, we analyze network synchronization to determine if the network complies with the ideal world specifications. To preserve the common prefix property, the inter-arrival time between two blocks must be long enough to allow each node to have an up-to-date blockchain [44]. In other words, when a new block is released, all nodes must have the previous block in their blockchain. If a node does not exhibit this behavior (*i.e.* due to propagation delay), it is vulnerable to partitioning. To concretely evaluate the network synchronization, we use **Heuristic 2** below.

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

To elaborate **Heuristic 2**, we again refer to Fig. 7.3. When our crawler received the new block

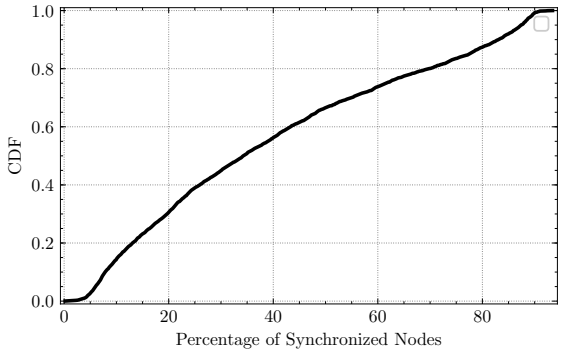


Figure 7.6: CDF of synchronized nodes reported from our measurements. Our results show that in ≈9.98 minutes on average, only 39.43% nodes have the latest block. The results are far from the ideal world specifications, indicating a block high propagation delay.

570367 (b_{i+1}) from the mining node, the "synced_blocks" value was 570366 (b_i). This means that before sending the new block, the mining node's blockchain tip was 570366 ($\mathcal{C} = b_{i-1}$). Hence, the mining node had an up-to-date blockchain. Similarly, at the time of receiving the new block 570367, the crawler expected all its connected peers to have the "synced_blocks" value 570366 in order to satisfy the synchronization property [44]. If the value of "synced_blocks" is less than 570366 (*i.e.* 570365), the peer is behind the blockchain by one block. The "synced_blocks" value of all connected peers was obtained from the RPC API. We sampled this information every time we received a block from a mining node. As a result, we were able to measure the percentage of synchronized nodes with an up-to-date blockchain.

We plot the CDF of synchronized nodes in Fig. 7.6, showing a weak network synchronization. We observed that the average block time was ≈9.98 minutes during which only 39.43% nodes had the up-to-date blockchain. Moreover, compared to the measurements in prior works [34, 92], our results indicate that the network synchronization is deteriorating over time.

7.5.1 Bitcoin Network Asynchrony

Since the overall network suffers from a weak synchronization, it is therefore intuitive to assume that the mining nodes may also experience a high latency in the block reception as well. Moreover,

non-uniform delay in the block propagation indicates an asynchronous network. To the best of our knowledge, the notable work that closely captures this behavior is conducted by Pass *et al.* [80] in which they performed a theoretical analysis to understand the performance of Bitcoin in the *non-lock-step* synchronous execution. This execution model allows an adversary to delay the block by a parameter Δ, giving the adversary a head start mining advantage. Pass *et al.* [80] assumed that after Δ, all the mining nodes simultaneously receive a block to start the next round. More precisely, they assumed that all honest miners *"freeze"* and do not start mining until all miners receive the block. In practice, however, when miners (honest or dishonest) receive a block, they immediately start mining the next block [76]. Therefore, if miners receive blocks at different times and start mining immediately, the network becomes asynchronous rather than *non-lock-step* synchronous. This hypothesis can only be validated by experimentally observing the block propagation among mining nodes, which we do in the following.

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

In our second experiment, we made three key observations. First, we noticed that the mining nodes received blocks faster than the non-mining nodes. On average, all mining nodes received a block within one minute. Second, we also noticed that the mining nodes received blocks at different time intervals, confirming asynchronous execution. Third, we observed that the block propagation

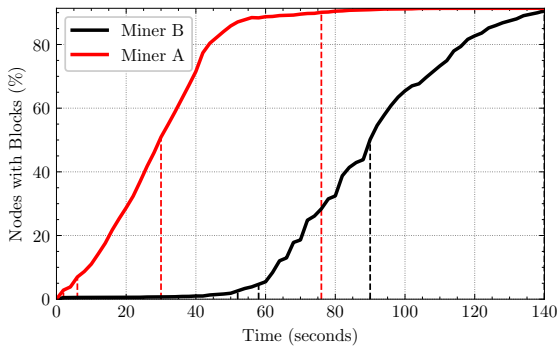


Figure 7.7: Block propagation of two miners, sampled at one second interval. Note that Miner A's block reaches $|M|/2$, $|M|$, $|\mathbb{N}|/2$, and $|\mathbb{N}|$ at 2, 6, 30, and 76 seconds, respectively. In contrast, Miner B's block reaches the same set of nodes at 52, 58, 90, and 140 seconds respectively. Miner A has a significant advantage over Miner B in terms of block propagation.

delay for each mining node varied, demonstrating variations in their network reachability (*i.e.* the number of connected peers). Variations in block propagation delay confirms that the mining nodes do not form a completely connected $M \times M$ topology.

To further highlight the aforementioned observations, we present an example from our dataset in Fig. 7.7, showing block propagation for two mining nodes randomly selected from the second experiment. For simplicity, we label the two miners as "Miner A" and "Miner B." Fig. 7.7 shows that when "Miner A" released the block, within 2 seconds, the block reached half of the mining nodes, and within 6 seconds, the block reached all the mining nodes. Moreover, within 76 seconds, the block reached $\approx$90% of all the nodes. In contrast, when 'Miner B" released the block, the block took 52 seconds to reach half of the mining nodes and 58 seconds to reach all the mining nodes. The block took 140 seconds to reach $\approx$90% network.

From Fig. 7.7, we derived the following conclusions. (1) Since mining nodes receive blocks at different times, therefore, the execution of the Nakamoto protocol in Bitcoin is asynchronous instead of *lock-step* synchronous [44] or *non-lock-step* synchronous [80, 83]. (2) Logically, this behavior suggests that the mining nodes do not form a completely connected $M \times M$ topology. This is analogous to our illustration in Fig. 7.2, where Mining Pool A is two hops away from Mining Pool

B, and Mining Pool C is one hop away from Mining Pool B. If Mining Pool B broadcasts a block through its mining node, Mining Pool C is likely to receive it before Mining Pool A. (3) Variations in the delay suggest that the Bitcoin mining nodes have a varying network reachability.

7.6 The *HashSplit* Attack

Nakamoto's consensus in a *non-lock-step* synchronous network increases the fork probability, wastes the effort of the honest miners, and lowers the cost for the majority attack [34, 80, 92]. Moreover, as indicated by Pass *et al.* [80], the problem becomes worse if the network is fully asynchronous, thus allowing an adversary to mount new attacks to violate the blockchain *safety* properties. Since our measurements clearly show that the Bitcoin network is fully 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 a high probability. Towards this objective, we present *HashSplit* which allows an adversary to exploit the asynchrony and orchestrate concurrent mining on multiple branches of a public chain to violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

HashSplit is a lower bound construction that shows: (1) an adversary with an arbitrary hashing power can violate $\mathcal{Q}_{cq}$, (2) an adversary with 26% hash rate can violate both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ with a high probability, and (3) the requirement for a majority attack under any hash rate can be amortized for all computationally admissible Bitcoin executions. From our measurements (§7.3–§7.5.1), we note that the asynchronous network creates a natural partitioning among the mining nodes which then expands the strategy space for an adversary to launch various attacks when combined with the Bitcoin mining policies [76, 80, 39, 34, 44].

It is worth noting that, unlike the balance attack [77] or the routing attacks [5, 92], *HashSplit* does not require the adversary to disrupt the network communication through BGP hijacking. Instead, the adversary simply exploits the existing propagation delay among the mining nodes and the natural partitioning created by the asynchronous network to split the hash rate. Below, we present the threat model and the attack procedure for the *HashSplit* attack.

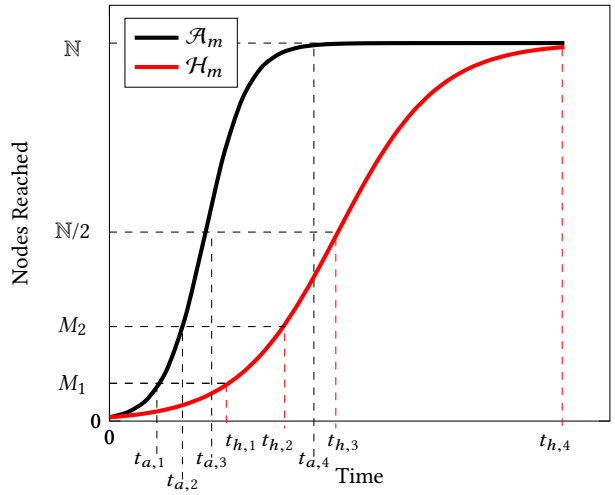


Figure 7.8: Generalized illustration of Fig. 7.7. For simplicity of modelling, we assume a uniform hashing power distribution in M (*i.e.* $M_1 \approx |M|/2 \approx 51\%$ hash rate and *i.e.* $M_2 \approx |M|/2 \approx 49\%$ hash rate). $\mathcal{A}_m$ is well connected compared to $\mathcal{H}_m$. If $\mathcal{H}_m$ and $\mathcal{A}_m$ concurrently produce a block, $\mathcal{A}_m$ wins race due to $\mathcal{H}_m$'s propagation delay. Here $t_{a,1}, t_{a,2}, t_{a,3}$ and $t_{a,4}$ are times when $\mathcal{A}_m$'s block reaches 50% miners, 100% miners, 50% network, and 100% network. Accordingly, $t_{h,1}...t_{h,4}$ are the corresponding values for $\mathcal{H}_m$.

7.6.1 Threat Model and Attack Objectives

For *HashSplit*, we assume an adversary $\mathcal{A}_m \in M$ with less than 51% hash rate. $\mathcal{A}_m$ follows the experiment methodology in §7.3–§7.5 to connect to all $P_i \in \mathbb{N}$, identify the mining nodes M, estimate their hashing power using the block mining rate (Fig. 7.5), and obtain the block propagation pattern of each mining node (Fig. 7.7). After identifying all $P_i \in M$, $\mathcal{A}_m$ maintains a direct connection with them to instantly send or receive blocks. Using the measurement methodology in Fig. 7.7, $\mathcal{A}_m$ calculates how a block generated by each $P_i \in M$ reaches all $P_i \in \mathbb{N}$. If $\mathcal{A}_m$ samples the block propagation pattern of each $P_i \in M$, then Fig. 7.7 can be expressed in terms of a general block propagation model in Fig. 7.8.

Fig. 7.8 shows that $\mathcal{A}_m$ has a good network reachability like Miner A in Fig. 7.7, while $\mathcal{H}_m$ (an honest miner) has poor network reachability like Miner B in Fig. 7.7. Since Fig. 7.7 is sampled at one second interval, $\mathcal{A}_m$ knows precisely (in seconds) at what time each $P_i \in \mathbb{N}$ receives a block.

By calculating the difference in the block 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 [76] and is currently deployed in all Bitcoin Core versions. Finally, we assume that (1) $\mathcal{A}_m$ cannot influence the communication model of other $P_i \in \mathbb{N}$ by launching routing attacks [5, 77], and (2) there is no other attack (*i.e.* selfish mining [96]) 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 [5, 34, 80].

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 [76]. In *HashSplit*, $\mathcal{A}_m$ achieves these goals by exploiting block propagation delay 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. In a perfect split, $\mathcal{A}_m$ splits the network hash rate into $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$ ($\alpha + \beta = 1$), and mines on the branch with a 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 a high probability.

7.6.2 Attack Procedure

7.6.2.1 Identifying Vulnerable Nodes

To split the blockchain, $\mathcal{A}_m$ first identifies the vulnerable mining nodes with a high block propagation delay by running algorithm 5. In algorithm 5, $T_{a,j}$ and $T_{i,j}$ are *reachability times* for $\mathcal{A}_m$ and other $P_i \in M$, respectively. $\mathcal{A}_m$ initializes four lists (aList...dList) and four variables (aMax...dMax). For each $P_i \in M$, $\mathcal{A}_m$ computes the time windows $\delta_1 ... \delta_4$ that represent the difference between the block propagation time of $\mathcal{A}_m$ and the target mining node. For intuition, we again refer to Fig. 7.7, in which if assume Miner A as $\mathcal{A}_m$ and Miner B as $\mathcal{H}_m$, then algorithm 5 outputs $\delta_1 = 50$, $\delta_2 = 52$, $\delta_3 = 60$, and $\delta_4 = 64$ seconds, respectively. Therefore, algorithm 5 provides the difference in the *reachability time* of all $P_i \in M$ relative to $\mathcal{A}_m$'s *reachability time*. Additionally, algorithm 5 also determines the most vulnerable node with the maximum *reachability time* difference, which can be the easiest target to initiate the split.

7.6.2.2 Blockchain Splitting

After discovering the vulnerable nodes, $\mathcal{A}_m$ splits the blockchain into two branches, $\mathcal{C}_1$ and $\mathcal{C}_2$, and miners into two groups, M_1 and M_2, using algorithm 6. We define the combined hash rate of M_1 as α and M_2 as β. algorithm 6 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 5), $\mathcal{A}_m$ 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 [76]). 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 Fig. 7.8), $\mathcal{A}_m$ violates the ideal functionality

Algorithm 5: Identifying Vulnerable Mining Nodes

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

Algorithm 6: 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 $t_{i,2}$ (Fig. 7.8).
Case 2: Any $P_i \in M$ finds $b^{'}_{r+1}$ before $\mathcal{A}_m$
Strategy 2: $\mathcal{A}_m$ violates the ideal functionality (see **onStart** in Fig. 7.1) 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.

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 6 provides two strategies for the adversary to split the blockchain ledger into two branches.

Perfect Split. As described in §7.6.1, the perfect split leads to $\mathcal{C}_1 \leftarrow \alpha = 0.51$ and $\mathcal{C}_2 \leftarrow \beta = 0.49$. If $\mathcal{A}_m$, with a hash rate α_1, mines on $\mathcal{C}_1$, we define the combined hash rate of all miners in M_1 as $\alpha = \alpha_1 + \alpha_2$. $\mathcal{A}_m$ can achieve the perfect split since it knows the block propagation pattern and the hash rate distribution (Fig. 7.8) of all the miners. $\mathcal{A}_m$ can time both strategies in algorithm 6 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 6, (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 Miner A in Fig. 7.7 and all other miners have block propagation patterns of Miner B in Fig. 7.7. 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.

7.6.2.3 Block Race

Once the perfect split is achieved, the chains $\mathcal{C}_1$ and $\mathcal{C}_2$, enter in a block race. To formally analyze the race conditions, we revisit the mathematical underpinnings of the Nakamoto consensus.

Bitcoin mining can be modeled as a Poisson process with inter-block times exponentially distributed with mean $\tau = 600$ seconds. A valid block has the double hash of the block header less than the difficulty SHA256(SHA256((Header))$< d \in [0, 2^{256} - 1]$. On average, a miner computes $m = 2^{256}/d$ hashes to mine a block [52]. 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 [52]. When the hash rate is split into α and β (algorithm 6), 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 6, 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 $t_o/(t_o + t_o^{'}) = \alpha/(\alpha + \beta)$ [52, 53, 88]. 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 other two miners h_1 and h_2 on the second branch $\mathcal{C}_2$.

After executing algorithm 6, $\mathcal{A}_m$ needs to maintain the fork for k consecutive blocks to violate

Block Race

Fork Persists:

- $\mathbf{f_1}$: $\mathcal{A}_m$ produces a block on C_1. No other miner produces a block on either C_1 or 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 C_1 and either h_2 or h_3 produce a block on C_2. $\mathcal{A}_m$ sends its block to h_1 who mines on C_1. h_2 and h_3 mine on 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 C_1 and both h_2 and h_3 produce a block on C_2. Three chains appear C_1, C_2, and C_3. $\mathcal{A}_m$ sends its block to h_1 and both mine on C_1. h_2 and h_3 mine on 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 C_1 and no miner on C_2 produces a block. $\mathcal{A}_m$ sends block to h_2 to maintains the perfect split. Probability is $\alpha_1\alpha_2(1-\beta_1)(1-\beta_2)$.
- $\mathbf{f_5}$: h_1 produces a block on C_1 and either h_2 or h_3 produce a block on C_2. C_1 and C_2 persist (perfect split exists) and $\mathcal{A}_m$ mines on 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 C_1 and both h_2 or h_3 produce a block on C_2. Three chains form (C_1, C_2, C_3). $\mathcal{A}_m$ receives block from h_1 and both mine on C_1. h_2 and h_3 mine on 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 C_1 and either h_2, or h_3, or both produce blocks on C_2. Three or four branches can appear. $\mathcal{A}_m$ mines with h_1 to maintain the hash rate advantage. Event probability is $\alpha_1\alpha_2\beta_1(1-\beta_2) + \alpha_1\alpha_2\beta_2(1-\beta_1) + \alpha_1\alpha_2\beta_1\beta_2$.
- $\mathbf{f_8}$: Both h_2 and h_3 produce blocks on C_2 and no miner on C_1 produces a block. C_1 resolves and C_2 and C_3 form. $\mathcal{A}_m$ mines on h_2's branch for higher hash rate advantage. Event probability is $\beta_1\beta_2(1-\alpha_1)(1-\alpha_2)$.
- $\mathbf{f_9}$: No miner produces block on either C_1 or C_2. The original fork persists. Event probability is $(1-\alpha_1)(1-\alpha_2)(1-\beta_1)(1-\beta_2)$.

Fork Gets Resolved:

- $\mathbf{r_1}$: h_1 produces a block on C_1 before $\mathcal{A}_m$, and neither h_2 or h_3 produce a block on C_2. 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 C_1 produces a block. Fork gets resolved and $\mathcal{A}_m$ mines on h_2's branch to maintain the hash rate advantage. Event probability is $\beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1)$.

Figure 7.9: Block race after the adversary executes algorithm 6. For each event, we show the event probability and the adversary's strategy for the next round.

$\mathcal{Q}_{cp}$. 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 race conditions under which the *HashSplit* attack succeeds or fails:

1. If forks persist for more than k blocks, $\mathcal{Q}_{cp}$ is violated, and the attack succeeds partially.

2. If forks resolve before k blocks and $\mathcal{C}_1$ wins, $\mathcal{Q}_{cq}$ is violated. The attack succeeds partially.

3. If 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. The attack succeeds completely.

4. If forks persist for k blocks and get resolved at $k+1$ block, with all k blocks mined by $\mathcal{A}_m$, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated. Moreover, in that case, the *HashSplit* attack becomes a majority attack since the adversary mines all blocks. In a synchronous network, the probability this event is 0.08 with $\alpha_1 = 0.26$ [88].

5. If forks resolve before or after k blocks and $\mathcal{C}_2$ wins, $\mathcal{A}_m$ loses all blocks. The attack fails.

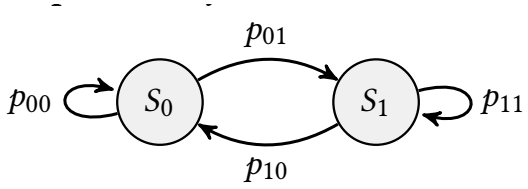


Figure 7.10: State machine representation of a block race. Transition probabilities are p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X = \mathbf{F}]$, $\mathbb{P}[X = \mathbf{R}]$, $\mathbb{P}[X = \mathbf{F}]$, and $\mathbb{P}[X = \mathbf{R}]$, respectively.

Clearly, *HashSplit* relies on the block race outcomes in which the forks persist or resolve. In Fig. 7.9, we formally analyze all outcomes of a 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 race outcome in Fig. 7.9. We also define **F** and **R** as the sum of events in which forks persist or resolve. In (7.3) and (7.4), we show the probability $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$.

$$\begin{aligned}
\mathbb{P}[X = \mathbf{F}] &= \alpha_1(1-\alpha_2)(1-\beta_1)(1-\beta_2) + \alpha_1\beta_1(1-\alpha_2)(1-\beta_2) + \alpha_1\beta_2(1-\alpha_2)(1-\beta_1) + \alpha_1\beta_1\beta_2(1-\alpha_2) \\
&\quad + \alpha_1\alpha_2(1-\beta_1) + (1-\beta_2) + \alpha_2\beta_1(1-\alpha_1)(1-\beta_2) + \alpha_2\beta_2(1-\alpha_1)(1-\beta_1) + \alpha_2\beta_1\beta_2(1-\alpha_1) + \alpha_1\alpha_2 \\
&\quad \beta_1(1-\beta_2)\alpha_1\alpha_2\beta_2(1-\beta_1) + \alpha_1\alpha_2\beta_1\beta_2 + \beta_1\beta_2(1-\alpha_1)(1-\alpha_2) + (1-\alpha_1)(1-\alpha_2) + (1-\beta_1)(1-\beta_2) \\
\mathbb{P}[X = \mathbf{F}] &= 3\alpha_1\alpha_2\beta_1\beta_2 - 2\alpha_1\alpha_2\beta_2 - 2\alpha_1\beta_1\beta_2 - 3\alpha_2\beta_1\beta_2 - 2\alpha_1\alpha_2\beta_1 + \alpha_1\beta_2 \\
&\quad + 2\alpha_2\beta_2 + 2\beta_1\beta_2 + \alpha_1\alpha_2 + \alpha_1\beta_1 + 2\alpha_2\beta_1 - \beta_2 - \alpha_2 - \beta_1 + 1
\end{aligned} \tag{7.3}$$

$$\begin{aligned}
\mathbb{P}[X = \mathbf{R}] &= \alpha_2(1-\alpha_1)(1-\beta_1)(1-\beta_2) + \beta_1(1-\alpha_1)(1-\alpha_2)(1-\beta_2) + \beta_2(1-\alpha_1)(1-\alpha_2)(1-\beta_1) \\
\mathbb{P}[X = \mathbf{R}] &= 2\alpha_1\alpha_2\beta_1 + 2\alpha_1\alpha_2\beta_2 + 2\alpha_1\beta_1\beta_2 - 3\alpha_1\alpha_2\beta_1\beta_2 + 3\alpha_2\beta_1\beta_2 \\
&\quad - \alpha_1\alpha_2 - \alpha_1\beta_1 - \alpha_1\beta_2 - 2\alpha_2\beta_1 - 2\alpha_2\beta_2 - 2\beta_1\beta_2 + \alpha_2 + \beta_1 + \beta_2
\end{aligned} \tag{7.4}$$

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, we make the following conclusions.

1. With algorithm 6 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.[2]

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 Fig. 7.9 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 $\mathbb{P}[X_k = \mathbf{R}]$ [39, 69]. Fig. 7.10 presents a state machine with S_0 and S_1 denoting states of forks and no forks, respectively. The transition probabilities for the state machine p_{00}, p_{01}, p_{10}, and p_{11} are $\mathbb{P}[X = \mathbf{F}]$, $\mathbb{P}[X = \mathbf{R}]$, $\mathbb{P}[X = \mathbf{F}]$, and $\mathbb{P}[X = \mathbf{F}]$, respectively.

6. Using Fig. 7.10 and incorporating block propagation delay from our measurements, we can compute the long term probability of a forked blockchain that violates $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$.

Incorporating Propagation Delay Advantage. Before computing the stationary distribution of Fig. 7.10, it is important to incorporate $\mathcal{A}_m$'s mining advantage due to the block propagation delay and block withholding. For instance, in $\mathbf{f}_1$, when $\mathcal{A}_m$ produces a block and withholds until h_2 or h_3 produce blocks, $\mathcal{A}_m$ can leverage the waiting time and the block propagation time to extend the newly mined block. The gap between $t_{a,1}$ and $t_{h,1}$ (or $t_{a,2}$ and $t_{h,2}$) provides additional time for $\mathcal{A}_m$ to mine the next block. To model this advantage, we first need to characterize the effect of block propagation delay on each miner's hash rate. Let $t_{a,0}$, $t_{h_{1,0}}$, $t_{h_{2,0}}$, $t_{h_{3,0}}$ be times at which $\mathcal{A}_m$, 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 block propagation delay and the hash rate can be obtained as:

[2]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).

$$\alpha_1 = \frac{\tau}{t_{a,0}}, \alpha_2 = \frac{\tau}{t_{h_1,0}}, \beta_1 = \frac{\tau}{t_{h_2,0}}, \beta_2 = \frac{\tau}{t_{h_3,0}} \tag{7.5}$$

$$\alpha_1 = \frac{\tau}{t_{a,0} + t_{a,1}},\ \alpha_2 = \frac{\tau}{t_{h_1,0} + t_{h,1}},\ \beta_1 = \frac{\tau}{t_{h_2,0} + t_{h,1}},\ \beta_2 = \frac{\tau}{t_{h_3,0} + t_{h,1}} \tag{7.6}$$

Considering $\alpha_1 = 0.26$, $\alpha_2 = 0.25$, $\beta_1 = 0.25$, $\beta_2 = 0.24$, and $\tau = 600$ seconds, from (7.5), $t_{a,0}$, $t_{h_1,0}$, $t_{h_2,0}$ become 2308, 2400, 2400, and 2500, respectively. Plugging these values in (7.6), the hash rate of each miner becomes $\alpha_1 = 0.259$, $\alpha_2 = 0.244$, $\beta_1 = 0.244$, and $\beta_2 = 0.235$. Next, to incorporate $\mathcal{A}_m$'s advantage in a block race, we convert δ_1 in algorithm 5 as the mining advantage that increases α_1. In our model, δ_1 with respect to all honest miners is $52 - 2 = 50$ seconds. In 50 seconds, $\mathcal{A}_m$ gets $(\delta_1/\tau = 0.0833$ fraction of additional mining power. As a result, the *effective hash rate* of each miner becomes $\alpha_1 = 0.3423, \alpha_2 = 0.2163, \beta_1 = 0.2163$, and $\beta_2 = 0.2073$. Moreover, $\mathbb{P}[X = \mathbf{F}]$ and $\mathbb{P}[X = \mathbf{R}]$ become 0.739 and 0.261, respectively.

This advantage can be extended to miners when they resolve forks ($\mathbf{r}_1$ and $\mathbf{r}_2$ in Fig. 7.9). If a fork resolves, the probability that it appears in the next round will be less than 0.739. The winning miner will have $t_{a,1}$ advantage over $\mathcal{A}_m$, and $t_{h,1}$ advantage over other miners. Empirically, $t_{a,1}$ accounts for $2/600 = 0.0033$, and $t_{h,1}$ accounts for $52/600 = 0.087$ fraction of the mining power. Therefore, if a fork resolve, the probability that it appears in the next round becomes $\mathbb{P}[X = \mathbf{F}] = 0.683$. Using these values, we can construct the transition probability matrix for Fig. 7.10.

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

In (7.7), we derive the stationary distribution of P to calculate the long term probability of a forked blockchain. The stationary distribution of P is a row vector π such that $\pi P = \pi$.

$$0.739\pi_1 + 0.261\pi_2 = \pi_1,\ 0.683\pi_1 + 0.317\pi_2 = \pi_2,\ \pi_1 + \pi_2 = 1 \tag{7.7}$$

From (7.7), $\pi_1 = 0.724$ and $\pi_2 = 0.276$, and the long term probability of a forked chain is significantly greater than of a single branch. Using the stationary distribution, we evaluate the impact of *HashSplit* on $\mathcal{Q}_{cp}$, $\mathcal{Q}_{cq}$, and the majority attack.

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

Chain Quality Property. Per (7.6), the block propagation affects the hash rate of each miner. As such, and even when not partitioning the blockchain, $\mathcal{A}_m$ can still mine more blocks that its hash rate allows. For instance, assuming an honest block race, and since $\delta_1 = 50$ seconds, $\mathcal{A}_m$ has $50/(3 \times 600)$ fraction of mining advantage over the other three miners ($\alpha_1 = 0.26, \alpha_2 = 0.223, \beta_1 = 0.223, \beta_2 = 0.213$). Moreover, if 100 blocks are mined, $\mathcal{A}_m$ will mine 28.29 blocks. From the ideal-world functionality view, $\mu - \mu^{'} = 2.29 \neq \epsilon$. $\mathcal{A}_m$ mines two blocks more than its hash rate, thus $\mathcal{Q}_{cq}$ is violated.

Common Prefix and Chain Quality. To violate $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$, a fork needs to persist or get resolved after k blocks, and $\mathcal{C}_1$ is the winning branch. Fig. 7.9 shows that $\mathbf{r}_2$ is the only outcome where forks get resolved to $\mathcal{C}_2$ with probability 0.2401. We analyze that by branching S_1 in Fig. 7.10 into two states and calculate the probability of $\mathcal{C}_2$ being the winning chain (computed as 0.167). Therefore, both $\mathcal{Q}_{cp}$ and $\mathcal{Q}_{cq}$ are violated with a probability of $1 - 0.167 = 0.833$.

Majority Attack. From Fig. 7.9, a majority attack happens if (1) $\mathcal{C}_1$ is the winning branch after k rounds, and (2) all blocks on $\mathcal{C}_1$ are mined by $\mathcal{A}_m$. This happens if for $k - 1$ rounds, one of the events $\mathbf{f}_i$, for $i = 1, 2, 3, 4, 7$ or 9 occurs, followed by event $\mathbf{f}_1$ on the k_{th} round. Similar to the analysis above, we can decompose this into a state machine where S_0 determines the probability of events $\mathbf{f}_i$, for $i = 1, 2, 3, 4, 7$ or 9, while S_1 determines the probability of $\mathbf{f}_i$ for $i = 5, 6$, or 8, and $\mathbf{r}_1$ or $\mathbf{r}_2$. From Fig. 7.9, we compute p_{00}, p_{01}, p_{10}, and p_{11} as 0.663, 0.337, 0.576, and 0.424, respectively. For $k = 6$, the result is $(0.63 \times 0.342) = 0.2156$. Therefore, with a probability of 0.2156, *HashSplit* allows $\mathcal{A}_m$ to launch a majority attack with only 26% hash rate. In the *lock-step*

synchronous or *non-lock-step* synchronous networks, the probability of successful majority attack with 26% hash rate is ≈ 0.08 [88].

In summary, *HashSplit* violates the blockchain *safety* properties with a high probability and significantly lowers the cost for the majority attack. In this chapter, we have only presented an attack against the mining nodes, although it can be launched against non-mining nodes (*i.e.* Bitcoin exchanges) to prevent them from generating $k-$confirmed transactions. As shown in §7.5, the non-mining nodes have a relatively poor network synchronization compared to the mining nodes, making them more vulnerable to *HashSplit*. Splitting the mining power to lower the cost of the 51% attack is known in the literature [77, 47, 5]. However, these attacks require an adversary to disrupt the communication model which can be detected by the victims. In contrast, the *HashSplit* adversary does not disrupt the communication, and only relies on the latency and mining policies to split the network. In the past five years, 26% hash rate has been possessed by various mining pools, including BTC.com, Antpool, and F2Pool (see Antpool's example [14]). All these features make *HashSplit* more practical, stealthy, and feasible in the current Bitcoin network. We acknowledge that the asynchronous network can be exploited in several other ways to launch new attacks similar to *HashSplit* or further refine *HashSplit* by incorporating new strategies. However, covering all those attacks is beyond the scope of this work.

7.7 Simulations and Results

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

adversary and the remaining 74% randomly assigned to the other five miners.

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

Fig. 7.11 shows that except for the 5th block, the adversary was able to find a block before any other miner in the network. After computing the block, the adversary waited for any other miner from the competing chain to release the block. In the meantime, it continued extending its own chain atop its previous block. In our results, we observed that at the 5th block, a miner on the second branch produced the block before the adversary. However, the adversary was able to mine the block immediately after, and it released the block to maintain the fork. Finally, at the 8th block, when the adversary mined its block, it did not withhold it. Instead, it released the block to

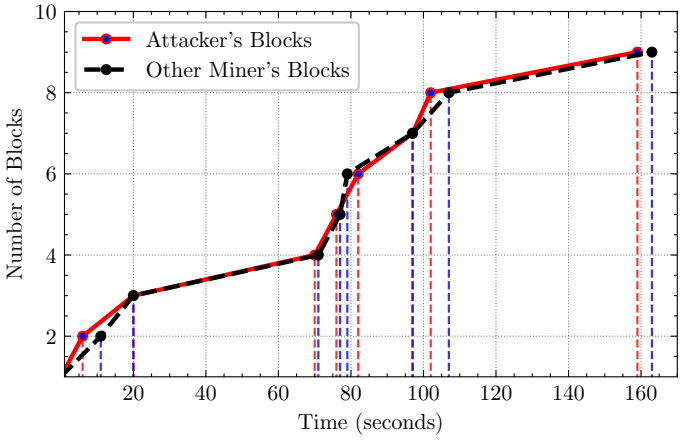


Figure 7.11: Simulations of the *HashSplit* attack. In each round (except 5th), the adversary with 26% hash rate is the first to produce a block and follows algorithm 6. In the 5th round, the adversary manages to produce the block before $t_{h,2}$. Adversary releases the chain after 8th block

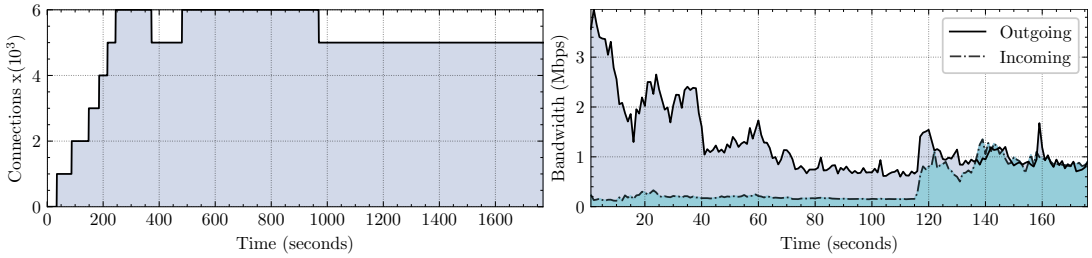
all miners in M, thereby forcing them to switch to the longer chain.

Our simulation results validate that by exploiting the asynchronous network, the adversary maintained two branches of the public chain to violate the common prefix property. The resulting chain had a majority of blocks mined by the adversary, which violated the chain quality.

7.8 Attack Countermeasures

In this section, we instead focus on attack countermeasures. Since *HashSplit* primarily exploits asynchronous network and block propagation delay, if $\delta_1 \ldots \delta_4$ in algorithm 5 are minimized, $\mathcal{A}_m$: (1) cannot split the mining nodes, and (2) cannot leverage a significant mining advantage from propagation delay. Additionally, if all $P_i \in M$ form $M \times M$ topology, Bitcoin will exhibit a *lock-step* or *non-lock-step* synchronous network, countering *HashSplit*.

To reduce propagation delay and form $M \times M$ network topology, we modified Bitcoin Core to allow fast connectivity among nodes. Existing Bitcoin Core suffers from poor network reachability, and a default node takes ≈120 days to connect to only 125 IP addresses (out of 6K–10K) [106], a long duration for a small fraction of the total network. To overcome this limitation, we modified Bitcoin Core to enable faster connectivity, as outlined in [3].



(a) Number of Bitcoin connections established in 1800 seconds

(b) Incoming and outgoing bandwidth consumption in MBps.

Figure 7.12: Performance evaluation of our Bitcoin Core version deployed on a full node. In less than 100 seconds, our node connected with over $6K$ reachable nodes while maintaining the bandwidth overhead under 6Mbps.

For performance evaluation, we deployed our Bitcoin Core client on a local machine and evaluated the connectivity speedup and bandwidth consumption, with results reported in Fig. 7.12. Our node connected with over $6K$ reachable nodes in less than 100 seconds, with a bandwidth consumption under 6Mbps (4Mbps incoming and 2Mbps outgoing) during the initial connectivity phase. Once the number of connections stabilize, the bandwidth consumption becomes ≈4Mbps. Our Bitcoin Core version is still in the testing phase and currently supports connections to IPv4 and IPv6 nodes.

From Fig. 7.12, we note that a Bitcoin node can connect to all IPv4 and IPv6 mining nodes in less than 100 seconds. Through direct connectivity and better reachability, the node can instantly receive blocks from honest mining nodes, thereby minimizing block delays and $\mathcal{A}_m$'s advantage. However, we acknowledge that $M \times M$ topology does not fully counter the attack. Our measurements show that the network latency can be heterogeneous such that two peers connected to a same node can have varying block propagation delay due to characteristics of the underlying Internet infrastructure (*i.e.* low bandwidth). Heterogeneous latency can be leveraged by $\mathcal{A}_m$ to launch the *HashSplit* attack even in $M \times M$ topology. Therefore, in addition to network layer remedies, we also require application layer defenses to counter the attack.

For application layer defenses, we equip our Bitcoin Core with a *fork resolution* mechanism. We note from Fig. 7.9, that the victim nodes have multiple branches of the same length in each round (*i.e.* $\mathcal{C}_1$ and $\mathcal{C}_2$) during the attack. Particularly, miners on $\mathcal{C}_1$ will continuously receive blocks

from $\mathcal{A}_m$, immediately followed by blocks from other honest miners. We leverage this sequence of block arrival to eliminate $\mathcal{A}_m$'s advantage and reduce the likelihood of a perfect split. In our Bitcoin Core [3], we provide a *fork resolution* mechanism in which a node removes the connection and bans the IP address for twenty four hours in the event of receiving $k = 6$ sequential blocks from it. This means that $\mathcal{A}_m$ will lose a direct connection to all mining nodes and will not be able to achieve a perfect split. $\mathcal{A}_m$ may deploy Sybil nodes in the network to connect to the victim. However, in that case $\mathcal{A}_m$ will lose δ_1 advantage over the victim since the block will be first relayed to the Sybil and then to victim node. Therefore, a combination of high network reachability and *fork resolution* mechanism can mitigate the *HashSplit* attack.

7.9 Summary

In this work, we show how the application-specific policies (*i.e.* time-based block precedence) and network layer constructs (*i.e.* asyncrony) in Bitcoin can be exploited to violate the fundamental blockchain properties. We present an ideal functionality the correctly captures the *modus operandi* of the Bitcoin network. By conducting large-scale measurements, 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 vulnerabilities. Moreover, our proposed countermeasures provide means to mitigate the attack by creating a *lock-step* synchronous network.

In §6, we noted that the Bitcoin security models do not fully characterize the impact of network asynchrony and network churn on the fundamental blockchain properties. In this chapter, by proposing and analyzing *HashSplit*, we have concretely investigated the impact of asynchrony on blockchain consistency. In the next chapter, we will demonstrate how an adversary can exploit the network churn to violate the blockchain consistency without using any mining power.

CHAPTER 8: SYNCATTACK: DOUBLE-SPENDING IN BITCOIN WITHOUT MINING POWER

In all the network layer attacks presented in prior chapters, we used the Nakamoto's attack construction as a blue print to model a block race between the adversary and the honest miners. Through measurements, we were able to uncover various discrepancies in the real world Bitcoin network that can be exploited to lower the cost of violating the blockchain consistency (*i.e.* 26% hash rate requirement in *HashSplit* §7). Despite significantly lowering the attack cost, our threat models assumed that the adversary controls some mining power to launch the attack and there is a stable mining power distribution during the attack. These limitations also exist in all prior works because the Nakamoto's attack construction stipulates that an adversary controls some mining power in order to violate the blockchain *consistency* property (*i.e.* through a double-spend attack). Moreover, as noted in [88], the adversary's success probability decreases exponentially if the hash rate distribution changes during the attack (*i.e.* new honest miners joining the network).

8.1 Motivation

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

Our findings are based on the two characteristics of the real world Bitcoin network that we observed in §5 and §6, and found that they have not been thoroughly explored in the past. First, we note that the blockchain forks, antecedent to a double-spend attack, do not solely rely on the hash rate distribution among the honest miners. In fact, forks can also occur due to a **weak network synchronization** that characterizes the blockchain view of each node in the network [92]. We further observe that network synchronization depends on the overlay topology of *reachable* nodes (e.g., using public IP addresses) in the network and the block propagation delay among those nodes.

If the overlay topology partitions or block propagation delay increase, the blockchain can fork even in the absence of an adversary. Therefore, network synchronization plays a key role in preserving the blockchain *consistency*. However, despite such a significance, network synchronization has not been comprehensively characterized in the Bitcoin security model.

Second, we note that prior security models [44, 80] ignored the *permissionless* nature of Bitcoin, which is intrinsic in its network design. The *permissionless* network allows nodes to leave or join the network at any time, thereby creating churn. Our study reveals that churn can be exploited to deteriorate network synchronization and launch new forms of partitioning attacks to disrupt the overlay topology. Our experiments also reveal a major vulnerability in Bitcoin Core that can be exploited to quickly partition the network by using only 28 IP addresses. More precisely, we show that by setting up 8 *reachable* nodes and 20 Docker containers, costing about $1,000 in total, an adversary can occupy the incoming connection slots of all the *reachable* nodes, preventing any new node from connecting to any of the existing nodes. This separation creates a partitioning between the incoming *reachable* nodes and the existing *reachable* nodes, which then increases with the network churn. The adversary exploits the partitioning to launch an attack and create forks by deteriorating the network synchronization. The adversary then uses those forks to launch a double-spend attack without using any mining power.

Contributions. In summary, our work makes a fundamental contribution by positioning network synchronization in the Bitcoin security model and analyzing its robustness in the *permissionless* settings. Additionally, by measuring and characterizing the behavior of the real world Bitcoin network, we propose SyncAttack, an attack that allows an adversary to deteriorate the network synchronization and launch a double-spend attack without mining power. Through this work, we conclude our attack surface analysis by presenting the most feasible attack on the Bitcoin network to date that violates the blockchain *consistency* while incurring a low cost of $1000. Our key contributions in this chapter are summarized as follows[1].

[1]For details on *reachable* nodes, *unreachable* nodes, and network synchronization, we refer to §6

- We present the first ideal functionality for the Bitcoin network synchronization. Our proposed ideal functionality characterizes the Bitcoin blockchain *consistency* property in light of the overlay network topology and the block propagation delay (§8.2).
- We conduct measurements to analyze the Bitcoin network synchronization in the real world network (§8.3). Our measurements reveal that, on average, only 52.2% *reachable* nodes have an up-to-date blockchain at any time, demonstrating weak network synchronization.
- We characterize the *permissionless* nature of the Bitcoin network by measuring churn among the *reachable* nodes (§8.3.2). Our measurements show a high network churn where 9% of the *reachable* nodes leave the network every day, replaced by almost an equal number of new *reachable* nodes. Additionally, we observe that all Bitcoin mining nodes experience churn.
- We show how the churn can be exploited to launch the SyncAttack in which the adversary partitions the network and creates forks by deteriorating the network synchronization (§8.4).
- We discover a vulnerability in Bitcoin Core that can be exploited to launch the SyncAttack by using only 28 unique IP addresses. We also demonstrate how SyncAttack can be used to (1) double-spend without using any mining power, and (2) reduce the *effective* mining power of the Bitcoin network. We also propose and implement the SyncAttack countermeasures.

.

8.2 Ideal Functionality for Bitcoin Network Synchronization

SyncAttack is a contrast between the Bitcoin ideal functionality and its real world implementation. Therefore, in keeping with the flow, we first present the ideal functionality below.

For the ideal functionality, we assume a set of *reachable* nodes N_r that execute the Nakamoto consensus for l rounds, arbitrated by a trusted party $\mathcal{F}_{\mathsf{syn}}$. Each $n_i \in N_r$ establishes eight outgoing

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

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

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

Receive: Consistent with the current Bitcoin protocol, if a node n_i receives two valid blocks $b_{r+1} \preceq b_r$ and $b'_{r+1} \preceq b_r$ in any round, where those blocks are linked to the same parent block b_r, n_i will stop its computation and start mining on the block that it receives the earliest. For instance, if $b_{r+1} \preceq b_r$ is received at t_1 and $b'_{r+1} \preceq b_r$ is received at t_2, where $(t_1 < t_2)$, then n_i mines on $b_{r+1} \preceq b_r$. Additionally, n_i forms two chains $\mathcal{C}_1 \leftarrow b_{r+1} \preceq b_r$ and $\mathcal{C}_2 \leftarrow b'_{r+1} \preceq b_r$, with $\mathcal{C}_1$ as the dominant chain on which n_i mines. Then, n_i, relays $b'_{r+1} \preceq b_r$ to O_i and I_i, and moves to the next round.

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

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

Figure 8.1: Ideal functionality for the Bitcoin network synchronization. The two conditions specified in the ideal functionality ensure that all *reachable* nodes in Bitcoin eventually receive a block and the maximum block propagation delay among the *reachable* nodes is bounded by a delay threshold parameter to prevent forks with a high probability

connections, making the average network outdegree $\deg^+(N_r)$, which then enables the block propagation in $k = \log_{\deg^+(N_r)} |N_r|$ steps. Each step adds a fixed propagation delay, t, and the network synchronizes if no fork appears in time kt while N_r maintains a minimum outdegree $\deg^+_{\min}(N_r)$. Fig. 8.1 provides the ideal functionality details, and Theorem 2 specifies bounds on $\deg^+(N_r)$ and block propagation delay that preserve the network synchronization.

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

Proof. For the proof sketch, we show that the proposed protocol in Fig. 8.1 securely realizes the

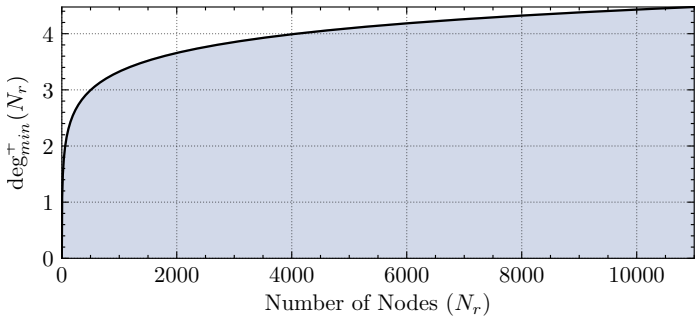


Figure 8.2: Relationship between $|N_r|$ and $\deg^+_{min}(N_r)$ required for a connected topology. In the current network size of ≈11K nodes [23], $\deg^+_{min}(N_r)$ must be greater than 4.47.

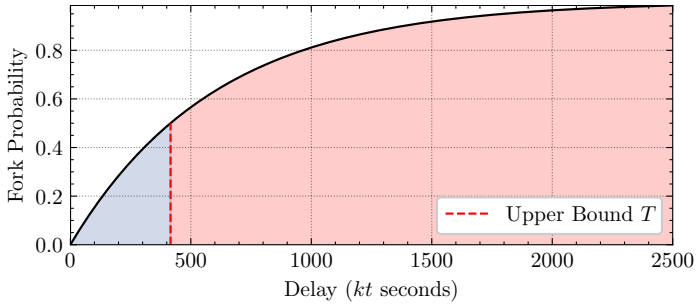


Figure 8.3: Fork probability due to block propagation delay kt. At kt=416 seconds, the fork probability becomes greater than 0.5. Therefore, we set the delay threshold T=416 seconds.

ideal functionality $\mathcal{F}_{\mathsf{syn}}$ by faithfully modelling the real world network characteristics [92, 23]. For that purpose, we specify the model parameters for the two conditions outlined in Theorem 2, and use values from the real world Bitcoin network [23].

The first condition in Theorem 2 $\left(\deg^+(N_r) \geq \deg^+_{\text{min}}(N_r)\right)$ refers to the Bitcoin network's capability of delivering blocks to all the *reachable* nodes. Therefore, $\deg^+_{\text{min}}(N_r)$ characterizes the minimum number of edges required to construct a connected overlay topology among the *reachable* nodes. Logically, if the network outdegree falls below the minimum network outdegree $\left(\deg^+(N_r) < \deg^+_{\text{min}}(N_r)\right)$, a group of *reachable* nodes will not be connected to the network, thus deteriorating the network synchronization [101].

To show that our ideal functionality satisfies the first condition in Theorem 2, we derive the mini-

mum Bitcoin network outdegree from [101], and compare it with the empirical values from the real world Bitcoin network [23]. From [101], we note that among $|N_r|$ *reachable* nodes, the minimum outdegree $\deg^+_{min}(N_r)$ is bounded by the following relationship.

$$\deg^+_{min}(N_r) \geq \left\lceil \frac{|N_r|}{|N_r|-1} \log_{O_i}(|N_r|) \right\rceil \tag{8.1}$$

Using (8.1), we plot $\deg^+_{min}(N_r)$ against $|N_r|$ in Fig. 8.2. We increase $|N_r|$ from 0 to 11K nodes, which is currently the number of *reachable* nodes in the network [23]. Fig. 8.2 shows that among $|N_r|$=11K nodes, if $\deg^+_{min}(N_r)$ is greater that 4.47 (*i.e.* 5), then there is a path from each node to every other node to deliver a block. Furthermore, through source code inspection, we observe that each *reachable* node in the real world network establishes eight outgoing connections (O_i=8), making that the network outdegree $\left(\deg^+(N_r) = 8\right)$ [18]. Since $\deg^+_{min}(N_r) < \deg^+(N_r)$, therefore, our ideal functionality satisfies the first condition in Theorem 2.

The second condition in Theorem 2 ($kt \leq T$) refers to the Bitcoin network's capability of preventing forks during block propagation. For that purpose, we specify that if the end-to-end block propagation delay kt is below the delay threshold parameter T, the probability of a fork remains below 0.5, thus preserving the blockchain *consistency* properties [44, 63].

In order to obtain a realistic value for T, we identify events during block propagation that can cause forks. Consider a node n_0 that mines a block $b_{r+1} \preceq b_r$ at time t_a. Next, consider another node $n_{|N_r|}$ as the last node in N_r to receive $b_{r+1} \preceq b_r$ at t_b. Therefore, the end-to-end delay kt becomes $t_b - t_a$, and a fork appears if $n_{|N_r|}$ mines $b'_{r+1} \preceq b_r$ between $t_b - t_a$. Let $\mathbb{P}[X = \mathbf{F}]$ be the probability that a fork appears during kt. [34] shows that $\mathbb{P}[X = \mathbf{F}]$ can be calculated as follows.

$$\mathbb{P}[X = \mathbf{F}] = 1 - (1-\lambda)^{kt} \tag{8.2}$$

In (8.2), λ is the probability of finding a block in 1 second. In Bitcoin, λ=1/600, where 600 is the average block time. Using λ=1/600, (8.2) can also be written as follows.

$$\mathbb{P}[X = \mathbf{F}] = 1 - \left(1 - \frac{1}{600}\right)^{kt} \tag{8.3}$$

In Fig. 8.3, we plot (8.3) by varying kt from 0 to 2500 seconds, and observe that $\mathbb{P}[X = \mathbf{F}]$ increases with kt. Since our objective is to keep $\mathbb{P}[X = \mathbf{F}]$ below 0.5, therefore, we can derive the cutoff value T=416 seconds that limits $\mathbb{P}[X = \mathbf{F}]$ below 0.5. Furthermore, given that $\deg^+(N_r)$=8, we can calculate the propagation delay t in each step k as $t = T/\log_{\deg^+(N_r)}|N_r|$ ≈32 seconds. Our bound on T is realistic since the prior work [34] reported $kt \approx 12$ seconds.

To summarize, our ideal functionality is admissible in the Bitcoin computation model since we show that (1) the average network outdegree is greater the minimum required outdegree, and (2) the realistic bound of $kt \leq 416$ prevents forks with a high probability. □

Compared to the existing theoretical frameworks [44, 80, 83], we make the following refinements in our ideal functionality to correctly model the real world characteristics of the Bitcoin network.

1. We acknowledge the default outgoing connection limits for a *reachable* node by setting $\deg^+(N_r)$ =8. In the prior works [44, 80], the authors assume that $N_r \times N_r$ is fully connected, which abstracts away the $\deg^+_{\text{min}}(N_r)$ requirement that the real world network must satisfy. Therefore, our ideal functionality captures the correct state of the overlay topology.

2. We note that forks that violate the blockchain *consistency* are not solely determined by the adversary's mining power as modeled previously in [44, 80]. Instead, if any of the two conditions in Theorem 2 are violated, forks will appear even in the absence of an adversary.

In summary, our ideal functionality embraces the reality of the real world overlay topology and introduces network synchronization in the Bitcoin security model. As such, by violating Theorem 2, an adversary can deteriorate the network synchronization to violate the blockchain *consistency* property through forks. In the following section, we conduct measurements to analyze how

closely the real world network follows the ideal functionality specifications.

8.3 Bitcoin Network Measurement

In this section, we present measurements to highlight the real world characteristics of the Bitcoin network. The key features we study are: (1) network synchronization in the real world, (2) blockchain forks and network outdegree due to variations in synchronization, (3) network churn caused by the *permissionless* network, and (4) partitioning possibilities due to churn.

For measurements, we collected data from an online service called Bitnodes that connects to all Bitcoin *reachable* nodes and reports their latest blockchain view after every five minutes [23]. We collected Bitnodes data from October 30, 2020 to December 30, 2020.

8.3.1 Bitcoin Network Synchronization

To analyze the network synchronization, we compare the latest block reported by Bitnodes with the latest block on the blockchain of all *reachable* nodes. Since Bitnodes crawlers are connected to all *reachable* nodes [92, 23], they instantly receive a newly mined block from any node. As such, Bitnodes' view of the network is similar to the view of $\mathcal{F}_{\mathsf{syn}}$ in Fig. 8.1. Taking that into account, we assume Bitnodes as $\mathcal{F}_{\mathsf{syn}}$ and apply **Heuristic 1** to analyze the network synchronization.

Heuristic 1. *When $\mathcal{F}_{syn}$ receives $b_{r+1} \preceq b_r$ from any $n_i \in N_r$, $\mathcal{F}_{syn}$ invokes* **Evaluate** *in* Fig. 8.1 *and counts the percentage of $|N_r|$ that report b_{r+1} on $\mathcal{C}$.*

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

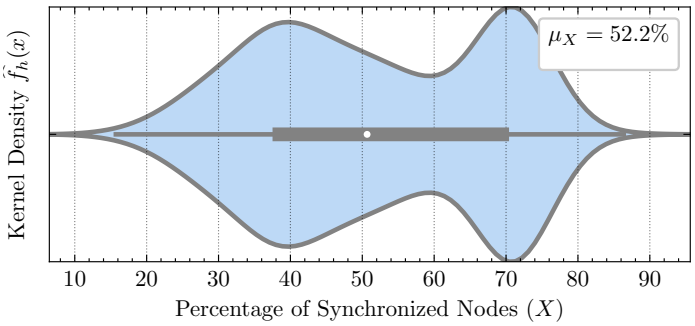


Figure 8.4: Results obtained by applying **Heuristic 1** on our dataset. Our results show a weak synchronization in the real world. On average, only 52.2% nodes had an up-to-date blockchain.

density estimation $\widehat{f}_h(x)$ of X [109] using the following formula.

$$\widehat{f}_h(x) = \frac{1}{z}\sum_{i=1}^{z} K_h\left(x - x_i\right) = \frac{1}{mh}\sum_{i=1}^{z} K\left(\frac{x - x_i}{h}\right) \tag{8.4}$$

In (8.4), K is the Gaussian kernel and h is the kernel bandwidth applied using the Scott's rule [100]. In Fig. 8.4, we plot $\widehat{f}_h(x)$ against X, showing that the average network synchronization is 52.2%, which is marginally above the 50% threshold specified in the ideal functionality. In other words, a few minutes after the release of a new block, only 52.2% nodes received that block, demonstrating weak synchronization. Our results are consistent with the prior work [92], where instances are reported with only 30% of nodes receiving a block even after ten minutes of its release.

Ideally, the network synchronization should be close to 100% so that all nodes share the same blockchain view. In 2013, the network synchronization was strong, since 90% of the *reachable* nodes received a block within 12 seconds [34]. However, since 2018, network synchronization appears to be deteriorating[92]. Moreover, we did not find any value of $x_i \in X$, where $\mathcal{N}_{\mathsf{syn}}$ was 100%, and the maximum and minimum values for $\mathcal{N}_{\mathsf{syn}}$ were 86.3% and 15.7%, respectively.

Due to this weak synchronization, it is logical to assume that the Bitcoin network has a high orphaned block rate due to forks. However, no orphaned blocks are reported by the network [25] in 2020, which is counter-intuitive given the values of $\mathcal{N}_{\mathsf{syn}}$ obtained in our measurements. It is therefore pertinent to explore *why the Bitcoin blockchain did not fork despite weak synchronization?*

Algorithm 7: Determining $\deg^{+}_{\text{min}}(N_r)$

Input: *reachable* nodes N_r
Blockchain $\mathcal{C} = (c_1, c_2, ..., c_z)$
Initialize Object O, O.**keys** $= N_r$
foreach $c_j \in \mathcal{C}$ **do**
    foreach $n_i \in N_r$ **do**
        if $c_j \in \mathcal{C}$ of $n_i = c_j$ **then**
            $O[n_i] \leftarrow 1$
        else
            $O[n_i] \leftarrow 0$
Return: O

8.3.1.1 Bitcoin Forks

By taking a closer look at the network anatomy [82], we discovered that the rarity of forks despite the weak synchronization is due to the mining centrality in the Bitcoin network [71, 108]. In the last few years, the Proof-of-Work (PoW) *difficulty* has significantly increased, allowing only a few miners with the sophisticated hardware to solve it [108, 64]. As a result, only a few nodes in the network mine blocks and release them to the other nodes, which then synchronize on those blocks [82]. Therefore, full nodes can be further categorized into (1) the mining nodes that mine blocks, and (2) the non-mining nodes that use those blocks to settle transactions.

The rarity of forks shows that the non-synchronized nodes in Fig. 8.4 are the non-mining nodes. Moreover, it also shows that $\deg^{+}(N_r)$ among the mining nodes is always greater than $\deg^{+}_{\text{min}}(N_r)$ and the block propagation delay among them is below kt seconds. Therefore, their behavior is consistent with the ideal functionality specification (Fig. 8.1), and they remain synchronized during each block round. Nevertheless, the rarity of the forks does not undermine the need for strong network synchronization even among the non-mining nodes. As stated earlier, the non-mining nodes use the blockchain to settle their transactions. As such, if the non-mining node are behind the blockchain as shown in Fig. 8.4, they are vulnerable to the temporal partitioning attack in which a malicious miner can corrupt their blockchain view [92].

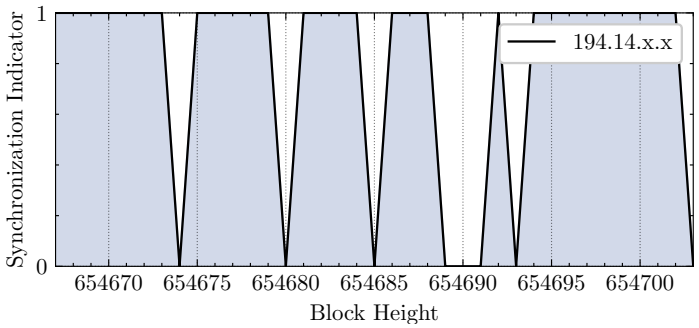


Figure 8.5: Network synchronization pattern of a node obtained from algorithm 7. When the node was synchronized, the corresponding value in the list was marked 1 (synchronization indicator). Therefore, the shaded region shows all the blocks for which the node remained synchronized.

8.3.1.2 Network Outdegree

Fig. 8.4 also shows a non-uniform width of $\widehat{f_h}(x)$ which indicates variations in $\deg^+(N_r)$. It is therefore worth investigating if the network outdegree falls below the minimum outdegree ($\deg^+(N_r) \leq \deg^+_{\text{min}}(N_r)$), thus preventing block delivery to a group of nodes for a long time.[2] A test case to determine this condition would be to find a non-synchronized node at a particular block and observe the node's synchronization pattern for all subsequent blocks. If the node stays behind the blockchain for all subsequent blocks, we can conclude that $\deg^+(N_r) \leq \deg^+_{\text{min}}(N_r)$, and there is no path in the overlay network that delivers blocks to that node.

In algorithm 7, we present our technique to determine if $\deg^+(N_r)$ is below $\deg^+_{\text{min}}(N_r)$ for a long time. We initialize an object O and set its keys as $n_i \in N_r$, and values as an empty list. We then iterate over each block $c_j \in \mathcal{C}$ and add 1 for a synchronized node and 0 otherwise. Finally, we output the object O and apply **Heuristic 2** to determine if $\deg^+(N_r)$ is below $\deg^+_{\text{min}}(N_r)$.

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

Heuristic 2 specifies that if a node was behind the chain in the past and eventually caught up, then

[2] *Bitnodes* does not relay blocks to its outgoing connections, thus 47.8% of its connections are non-synchronized.

there exists a path in the overlay network that delivers blocks to that node. Therefore, the average network outdegree is greater than the minimum network outdegree.

After applying algorithm 7 on our dataset, we did not find any *reachable* node that stayed behind the blockchain indefinitely. As an example, in Fig. 8.5, we plot the synchronization pattern of a *reachable* node for 30 consecutive blocks. We mask the last two octets of the node's IP address to preserve its privacy. Fig. 8.5 shows that each time the node was behind the blockchain, it eventually caught up and synchronized on the latest block.

Fig. 8.5 also shows variations in the synchronization pattern, indicating that block reception depends on the node's location in the overlay network relative to the mining nodes. For instance, for blocks 654670 and 654695, the node was found to be synchronized, suggesting a close proximity with the mining nodes of those blocks. On the other hand, for blocks 654675 and 654690, the node was distant from the mining nodes of those blocks, therefore it did not receive blocks even after a long time. Nevertheless, despite being behind the blockchain even for three consecutive blocks ($654690 - 654692$), the node eventually was synchronized.

These observations lead to two possible characterizations of the network outdegree. (1) The network outdegree is always greater than the minimum outdegree and the lack of synchronization is predominantly due to block propagation delay. (2) In the worst case assumption, even if the network outdegree becomes less than the minimum outdegree, it eventually recovers since the non-synchronized nodes eventually catch up with the blockchain.

Key Takeaways. From the synchronization analysis of the real world network, we make the following conclusions. First, the overall network synchronization is weak since only 52.2% nodes have an up-to-date blockchain at any time. However, because those nodes are non-mining nodes, the blockchain does not fork. Second, despite the weak synchronization, the network outdegree is usually greater than the minimum outedegree, which enables the non-synchronized nodes to catch up with the blockchain. Therefore, the research question then becomes *how can an adversary bring* $\deg^+(N_r)$ *below* $\deg^+_{\text{min}}(N_r)$ *for both mining and non-mining nodes in order to deteriorate*

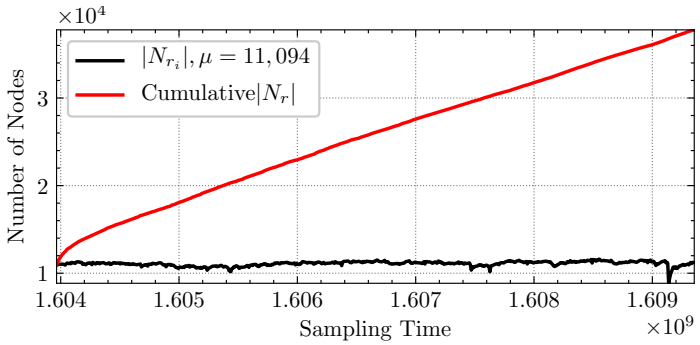


Figure 8.6: Cumulative number of *reachable* nodes and the average number of *reachable* nodes present in the Bitcoin network at any time. The gap between the two lines indicates a high network churn caused by the *permissionless* network.

the network synchronization and create forks? Our dataset reveals that an adversary can achieve this objective by exploiting the churn caused by the Bitcoin network's *permissionless* nature.

8.3.2 Bitcoin Network Churn

Since the Bitcoin network is *permissionless*, nodes can join or leave the network at any time [76]. The arrival and departure of nodes creates a churn and changes the network outdegree, which subsequently affects the block propagation and network synchronization. In the SyncAttack, the adversary exploits the churn to create a partitioning between the existing nodes and the arriving nodes, and use that partitioning to create forks and break the blockchain *consistency*. In this section, we analyze the Bitcoin network churn to extract useful insights for SyncAttack.

8.3.2.1 Measurement Results

Network Size. In 60 days, we collected 37,778 IP addresses of *reachable* nodes with ≈11,094 *reachable* nodes present in the network at any time. Fig. 6.6 shows the number of *reachable* nodes ($|N_r|$) present in the network at any time, as well as the cumulative number of unique IP addresses of nodes collected over the measurement duration. The growing gap between the two lines in Fig. 8.6 indicates a significant churn since the number of unique IP addresses increased contin-

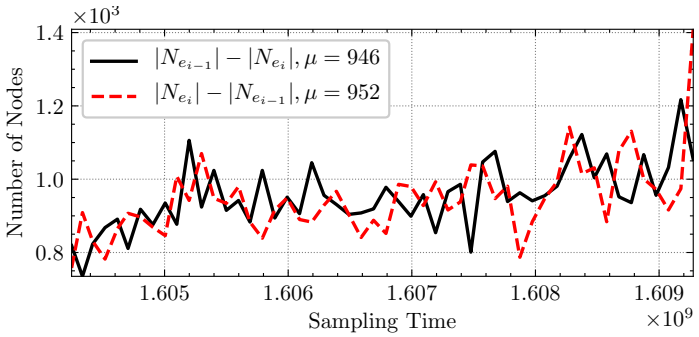


Figure 8.7: The number of arriving and departing nodes in the Bitcoin network. On average, in two months, 952 nodes joined and 946 nodes departed from the network every day.

uously while the number of nodes present in the network at any time remained constant (≈11K).

Arriving and Departing Nodes. After observing a high network churn, we then analyze the vulnerable network state created by the churn. When a *reachable* node departs from the network, all its connections are dropped, including the incoming connections from its peers. Those peers then try new outgoing connections to complete their default outgoing slots (8 in Bitcoin). If no other *reachable* node accepts their connection requests, the average network outdegree decreases, affecting the network synchronization (Fig. 8.1).

Similarly, if a node joins the network and no *reachable* node accepts its connections, the network outdegree remains low. Furthermore, if an adversary occupies all the node's incoming and outgoing connections, the node can be partitioned from the rest of the network [55]. Therefore, the node arrivals and departures create an imbalance in the overall network outdegree, which can be exploited by an adversary to split the network and control the communication model.

To analyze the number of arriving and departing nodes, we denote $N_{e_{i-1}} - N_{e_i}$ as the set of nodes present on the previous day $i-1$, and absent from the current day i. The resulting value $|N_{e_{i-1}}| - |N_{e_i}|$ gives the number of nodes that departed from the network on day i. Conversely, $|N_{e_i}| - |N_{e_{i-1}}|$ gives the number of arriving nodes that were not found on the previous day. In Fig. 8.7, we plot the number of departing nodes $|N_{e_{i-1}}| - |N_{e_i}|$ and the number of arriving nodes $|N_{e_i}| - |N_{e_{i-1}}|$ for 60 days. Our results show that, on average, 946 nodes departed from the network and 952 new nodes

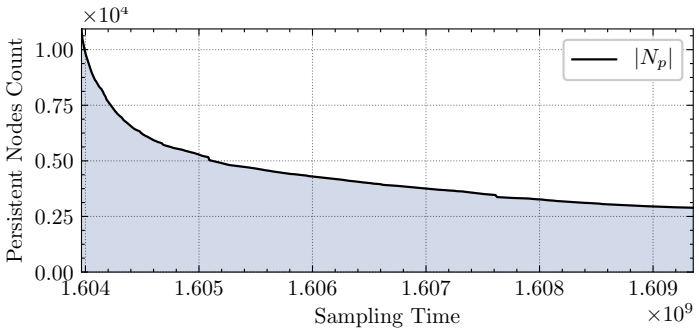


Figure 8.8: The number of persistent nodes R_p in the Bitcoin network. Note that over time, the curve flattens and we find 2,890 nodes that stayed persistently in the network.

joined the network every day. This shows a high churn and a high variation in the Bitcoin network outdegree, leading to a varying network synchronization observed in Fig. 8.4.

Note that if an adversary occupies all the incoming connections of the nodes that are already present in the network, the arriving nodes will not be able to establish connections to them. Additionally, if the adversary connects to the arriving nodes, then the network will be partitioned between the arriving nodes and the existing nodes in the network.

Persistent Nodes. Our measurements also revealed that despite churn, 2,890 nodes did not leave the network during the entire measurement study. For simplicity, we call them the "persistent nodes" (R_p), and plot them in Fig. 8.8 by counting the common elements in N_{e_i} and $N_{e_{i-1}}$. The key feature of the persistent nodes is that, unlike the arriving nodes, the outgoing connections of the persistent nodes cannot be easily controlled by an adversary [55, 54]. For instance, if a node $n_i \in R_p$ establishes all its outgoing connections to other nodes in R_p, those connections will not drop despite the departure of other *reachable* nodes that experience churn.[3] Therefore, the outgoing connections among R_p cannot be controlled by the SyncAttack adversary.

Mining Nodes. As mentioned in §8.3.1.1, the rarity of forks is due to the strong synchronization among the mining nodes. Detecting and partitioning the mining nodes allows the adversary to create forks and double-spend. As such, if the mining nodes are among the persistent nodes, the

[3]We assume that no $n_i \in R_p$ experiences link failures.

Algorithm 8: Detecting Mining Nodes

Input: *reachable* nodes N_r, Sampling rate γ
Blockchain $\mathcal{C} = (c_1, c_2, ..., c_z)$
Initialize Synced Nodes R_s, Counter q, Object M
foreach $c_j \in \mathcal{C}$ **do**
    foreach $n_i \in N_r$ **do**
        if $c_j \in \mathcal{C}$ of $n_i = c_j$ **and** $n_i \notin R_s$ **then**
            add n_i to R_s
    if $c_j \in \mathcal{C}$ of $n_i \neq c_j$ **and** $n_i \in R_s$ **then**
        remove n_i from R_s
Count q as the occurrence of each $n_i \in R_s$
Calculate node's lifetime $q \times \gamma$
Add $M[n[i]] = q \times \gamma$ **Return:** M

SyncAttack becomes less feasible since the adversary cannot control their outgoing connections to each other. In contrast, if the mining nodes experience churn, the SyncAttack becomes more feasible and the adversary can partition the mining nodes. Therefore, it is important to determine if the mining nodes experience churn and can therefore be targeted by the SyncAttack adversary.

To determine the churn among the mining nodes, we applied algorithm 8 to find all nodes with an up-to-date blockchain during their entire network lifetime. In algorithm 8, we sample all the synchronized nodes R_s for each block $c_j \in \mathcal{C}$. If any node in R_s is found to be present in the network and behind the blockchain, it is removed from R_s. Finally, we count the occurrence of each node in R_s, and represent it in an object M, with the object key representing the node's IP address and the object value showing the node's lifetime in the network while synchronized.

Our analysis reveals 18,077 nodes in the Bitcoin network that always have an up-to-date blockchain during their entire network lifetime. In Fig. 8.9, we plot the number of synchronized nodes against their network lifetime, showing that 72% of the synchronized nodes do not stay in the network for more than two days. Moreover, we found ≈5K nodes that stayed in the network for up to ≈11 days. It is logical to assume that the mining nodes are among those ≈5K synchronized nodes since a longer network lifetime is desirable for a mining node to prevent the synchronization overhead during the deployment of a new mining node. We did not find any synchronized node that stayed in the network for more than 11 days. Therefore, we conclude that the mining nodes are not among the persistent nodes, and churn make them vulnerable to the SyncAttack.

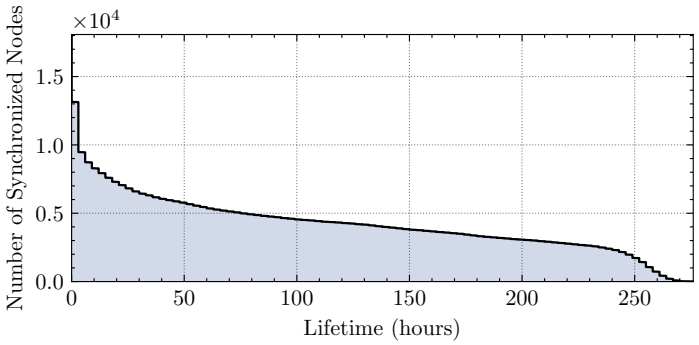


Figure 8.9: Network lifetime of synchronized nodes. Among the total 18,007 nodes, 72% nodes did not stay in the network for more than two days. Moreover, the maximum lifetime of a synchronized node was found to be ≈11 days. The results clearly show that all mining nodes experience churn.

Key Takeaways. From the churn analysis, we make the following key conclusions. First, the Bitcoin network has a high churn and ≈9% *reachable* nodes depart from the network every day, replaced by almost an equal number of arriving nodes.[4] A high churn also provides clues about the weak synchronization observed in Fig. 8.4. When nodes leave the network, the network outdegree decreases, which is then improved by the arriving nodes. However, the arriving nodes are usually behind the blockchain and it takes time to synchronize with the network. As a result, there are often behind the blockchain when Bitnodes queries them. Therefore, a high churn is another key factor behind the weak network synchronization observed in Fig. 8.4.

Second, we discovered (1) 2,890 persistent nodes that are always present in the network at all times, and (2) ≈5K synchronized nodes which include the mining nodes through which miners release their blocks in the network (see Fig. 7.2). We did not find an overlap between the persistent nodes and the mining nodes, leading us to a conclusion that all mining nodes experience churn.

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

8.4 The SyncAttack

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

8.4.1 Threat Model

For the SyncAttack threat model, we use the formalism introduced in §8.2 by specifying N_r *reachable* nodes in the network. Each $n_i \in N_r$ establishes O_i=8 outgoing connections and accepts I_i=117 incoming connections. Acknowledging the churn, we further divide N_r into N_i arriving nodes and N_e existing nodes. Prior to the SyncAttack, $|N_i|$=0 and $|N_e|$=11K.

Next, we assume an adversary $\mathcal{A}$ who runs A_r *reachable* nodes and A_u *unreachable* nodes. Each $a_i \in A_r$ maintains a Bitcoin blockchain and its source code is modified to allow more than 117 incoming connections from N_r. In contrast, no $a_i \in A_u$ maintains a blockchain or accepts incoming connections from N_r. Instead, each $a_i \in A_u$ executes a lightweight script that emulates the behavior of an *unreachable* node with only three functionalities defined below.

1. Establish an outgoing connection to $n_i \in N_r$ by simply performing the TCP handshake and exchanging the `VER` and `VERACK` messages [106].

2. In response to the GETADDR message, only relay the IP addresses of A_r to $n_i \in N_r$.

3. Optionally request the Bitcoin blocks from $n_i \in N_r$ and discard those blocks.

The above functionalities allow $a_i \in A_u$ to behave like an *unreachable* node without maintaining the ≈330GB blockchain. Moreover, by only relaying A_r in the ADDR messages, $\mathcal{A}$ ensures that the IP addresses of its *reachable* nodes reach the `new` table of each $n_i \in N_r$.

Number of Nodes required for SyncAttack. As stated earlier, $\mathcal{A}$ aims to occupy all the incoming and outgoing connections of N_i, and all the incoming connections of N_e. Therefore, it is important to estimate the number of the nodes required to achieve this objective in order to determine the attack feasibility. Since each $n_i \in N_r$ establishes O_i=8 outgoing connections, $\mathcal{A}$ needs to host only 8 *reachable* nodes to fill the outgoing connection slots of all the honest *reachable* nodes. Given that $|N_i| + |N_e|$ ≈11K at any time, each $a_i \in A_r$ needs to accept (11,000×8)/117≈753 incoming connections. This can be trivially achieved by modifying the Bitcoin Core source code and increasing the number of the incoming connections to 753 or more. The adversary can host its 8 *reachable* nodes on a cloud and assign a unique IP address to each node.

To occupy all of the incoming connections of $n_i \in N_r$, $\mathcal{A}$ needs to establish (117×11,000= 1,287,000) connections. Prior works on IP address-based partitioning attacks implicitly assumed that the Bitcoin nodes **only accepts one connection per IP address** [92]. Therefore, their threat model assumed a strong adversary (*i.e.* an ISP) that owns more than 100K IP addresses to target the Bitcoin network [106]. If we use the same model in the SyncAttack, $\mathcal{A}$ will need (1,287,000/8= 160,875) IP addresses to occupy all the incoming connections of the honest *reachable* nodes. With a modest estimate of $23 for acquiring an IP address [51] and $5 for hosting a virtual machine [79], the total cost for the attack exceeds $4 Million. However, in the following, we show that $\mathcal{A}$ can reduce the cost by exploiting a Bitcoin Core vulnerability.

Bitcoin Core Vulnerability. By inspecting the Bitcoin Core source code, we discovered a vulnerability that allows a single IP address to generate multiple outgoing connections to a *reachable* node. We found that instead of treating each IP address as a unique connection, a *reachable* node concatenates the IP address of an incoming connection with the port number and treats the result as a single connection. This functionality can be easily exploited to launch a denial-of-service attack

to occupy all the incoming connections of a *reachable* node. Considering the fact that there are 65,535 available ports on a commodity computer, $\mathcal{A}$ can easily occupy all 117 incoming connections of a node by using a single IP address and different port numbers for each connection.

To experimentally demonstrate this vulnerability, we set up a Bitcoin *reachable* node and developed a lightweight Bitcoin script that establishes outgoing connections to the *reachable* node. We observed that the script immediately occupied all the incoming connections of the *reachable* node using the same IP address and a different port for each connection. As a result, the *reachable* node was unable to accept any new connections from other *reachable* and *unreachable* nodes. In [2], we provide a video demonstration of our experiment and, due to ethical concerns, we will not release the script code. Moreover, we have patched the vulnerability in Bitcoin Core [2].

A single IP address can theoretically handle up to 65,535 incoming connections. Given 1,024 ports are reserved, $\mathcal{A}$ only requires (1,287,000/64,511 $\approx$20) machines with unique IP addresses. $\mathcal{A}$ can simply set up 20 Docker containers, each with a unique IP address and 64,511 ports in order to attack all the *reachable* nodes in the Bitcoin network. To summarize, the SyncAttack adversary only needs 8 commodity computers with a Bitcoin blockchain to use them as the *reachable* nodes, and 20 Docker containers with lightweight scripts to use them as the *unreachable* nodes. Using this approach, the attack cost can be reduced significantly from $4 Million to only $\approx$$800.

8.4.2 Attack Procedure

For the attack procedure, we assume that each $n_i \in N_r$ runs the default Bitcoin Core client with I_i=117 and O_i=8 connections. We later show in algorithm 9 that even if some nodes (*i.e.* Bitnodes) increase their connection limits, $\mathcal{A}$ can easily attack them by increasing the number of Docker containers with each container providing 64,511 new connections to $\mathcal{A}$.

For N_r *reachable* nodes in the network, we define $R_c = N_r \times 117$ as the total number of slots available for the *reachable* and *unreachable* nodes to occupy. Among those slots, we assume that R_o slots are already occupied by those nodes prior to the attack. Accordingly, we define

Algorithm 9: Occupying All Incoming Connections

Input: N_r, A_u
Compute: $C_a = |A_u| \times 64{,}511$
foreach $n_i \in N_r$ **do**
    if n_i accepts connection **and** $C_a > 1$ **then**
        connect to n_i **and** $C_a = C_a - 1$
    if n_i does not accept connection **and** $C_a > 1$ **then**
        $R_a = 0$ all slots occupied
    if $C_a = 0$ **then**
        add a Docker container and **connect** to n_i. $C_a = C_a + 64{,}511$
Return R_a=0

$R_a = R_c - R_o$ as the available slots that $\mathcal{A}$ occupies using lightweight scripts. When R_a=0, there is no available slot in the network for any new *reachable* or *unreachable* node. This is the focal point of the SyncAttack, since $\mathcal{A}$ causes a denial-of-service by ensuring that *no reachable node accepts any new incoming connection from other nodes in the network.* algorithm 9 describes how $\mathcal{A}$ occupies all the available slots. If any node accepts more than 117 incoming connections and $\mathcal{A}$'s connection slots are exhausted, then $\mathcal{A}$ adds a new container to its setup and ensures R_a=0.

When algorithm 9 is completed, no $n_i \in N_r$ can establish any outgoing connection to any other $n_j \in N_r$. However, $n_i \in N_r$ can establish an outgoing connection to any *reachable* node $a_i \in A_r$ controlled by $\mathcal{A}$, since those nodes still accept incoming connections. Once R_a=0 and the churn occurs, $\mathcal{A}$ starts to control the links between nodes in $|N_i|$ and $|N_e|$ to violate the ideal functionality specifications. In the following, we show $\mathcal{A}$'s strategies during churn.

Arriving Nodes. When a new node n_i joins the network for the first time, it queries a list of DNS seeds hardcoded in the *chainparams.cpp* file [18]. The DNS query returns a list of *reachable* addresses to which n_i establishes outgoing connections. After successfully connecting to a *reachable* node, n_i sends the GETADDR message to that node in order to receive an ADDR messsage in return containing up to 1000 IP addresses of other nodes in the network.

Since R_a=0 after algorithm 9, n_i can only establish an outgoing connection if the DNS seeds return an IP address of any $a_i \in A_r$. Once n_i connects to $a_i \in A_r$ and sends GETADDR message, $a_i \in A_r$ only returns the IP addresses of $\mathcal{A}$'s *reachable* nodes in A_r. Upon receiving those addresses, n_i makes all the eight outgoing connections to A_r.

When $\mathcal{A}$ learns the IP address of n_i from A_r, it forwards the address to A_u which then run algorithm 9 to occupy a_i's incoming connections. If n_i accepts incoming connections, algorithm 9 will ensure that all its incoming slots are occupied. As a result, all I_i and O_i of n_i are occupied by $\mathcal{A}$.

A key constraint in this attack procedure is that the DNS seeds must relay at least one IP address in A_r to n_i. To analyze how this can be achieved, we explored the DNS seed specification provided by a Bitcoin Core developer, which states that the DNS seeders *"return a good sample"* of *reachable* nodes in their response [33]. This means that the Bitcoin nodes owned by the DNS seed providers know a few IP addresses of *reachable* nodes in the network. Since $\mathcal{A}$'s nodes are connected to all the *reachable* nodes, $\mathcal{A}$ can send ADDR messages to its connections containing only the IP addresses in A_r. This procedure will increase the probability of IP addresses in A_r being relayed by the DNS seeders. $\mathcal{A}$ can also sidestep this process by parsing the publicly available DNS server database maintained by a Bitcoin Core developer [60]. The database records all the IP addresses that query the DNS seed. By constantly parsing the DNS database, $\mathcal{A}$ can learn the IP addresses of new nodes and target them without relying on the DNS query results.

Resource Optimization. In the SyncAttack, $\mathcal{A}$ only targets the *reachable* nodes due to their role in provisioning the network synchronization. It is possible that an arriving node is an *unreachable* node that does not accept incoming connections. In such a case, $\mathcal{A}$ can disconnect with that node by observing if the node drops all the incoming connections from $a_i \in A_u$.

Departing Nodes. When a *reachable* node departs from the network, its *reachable* connections will have one less outgoing connection that they have established with the departing node. When R_a=0, those nodes are unable to connect to any other node in N_r. However, if they have an IP address of any $a_i \in A_r$ in their `new` or `tried` tables, they eventually establish an outgoing connection with A_r to complete their outgoing slots. If the departing node rejoins the network at any time, it skips the DNS querying phase and attempts connections from its `new` and `tried` tables. If 11 seconds elapse without a successful connection, the node queries the DNS seeders [18]. If R_a=0, the node eventually connects to A_r based on the procedure described in the previous section.

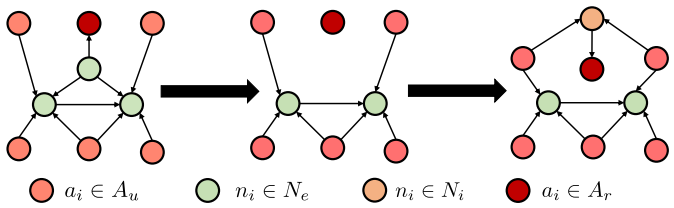


Figure 8.10: SyncAttack illustration showing how $\mathcal{A}$ occupies all the connections of the arriving nodes N_i and the outgoing slots of N_e, left opened by the departure of an existing node.

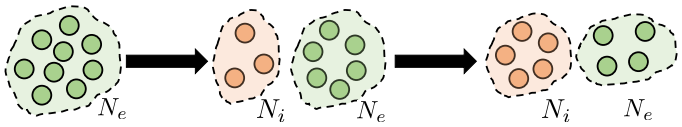


Figure 8.11: Due to churn, the size of N_e decreases and the size of N_i increases with time.

Network Partitioning. By maintaining R_a=0, $\mathcal{A}$ ensures that all the incoming and outgoing connections of N_i are established with A_u and A_r, respectively. Moreover, when any $n_i \in N_e$ departs from the network, its *reachable* connections only connect with A_r. Fig. 8.10 illustrates the network state when a node departs and a new node joins the network. Since no node in N_i can connect to any node in N_e (R_a=0), the network is partitioned between N_i and N_e. Moreover, the size of $|N_e|$ decreases and the size of $|N_i|$ increases with the churn. Fig. 8.11 illustrates the change in $|N_e|$ and $|N_i|$ due to churn. From Fig. 8.8, we note that it takes ≈52 days to flatten the curve, from which we obtained R_p=2,890 nodes. Therefore, the size $|N_i|$ will become $|N_e| - |R_p|$ in 52 days.

Communication Model. We now examine the communication model of the network N_i under churn and evaluate its compliance with the ideal functionality specifications in Fig. 8.1.

Since $\mathcal{A}$ controls all the incoming and outgoing connections of each $n_i \in N_i$, $\deg^+(N_i)$ becomes 0 (*i.e.* no edge between the honest nodes). This allows $\mathcal{A}$ to violate the first condition in Theorem 2, since $\deg^+(N_i)$ remains 0 despite the increasing network size. Additionally, by controlling all connections in N_i, $\mathcal{A}$ can delay the block propagation among nodes in N_i by more than kt seconds, violating the second condition in Theorem 2. This shows that when algorithm 9 is followed by

churn, the network is partitioned and $\mathcal{A}$ completely deteriorates synchronization in N_i.

We further notice that the current Bitcoin network is highly vulnerable to the SyncAttack since all mining nodes experience churn (§8.3.2). As a result, each mining node moves from N_e to N_i, which allows $\mathcal{A}$ to control all the incoming and outgoing connections of the Bitcoin mining nodes. $\mathcal{A}$ can then orchestrate a block race among miners by controlling the block propagation among them and allowing forks to appear. In the following, we show how $\mathcal{A}$ achieves that to launch a double-spend attack without using the mining power.

8.4.2.1 Double-spending in the SyncAttack

Given that (1) $\mathcal{A}$ completely controls the communication in N_i, and (2) all miners eventually become part of N_i, $\mathcal{A}$ can simply stop the block propagation among the mining nodes so that each miner extends their own chain. Such an attack will violate the common prefix property of the Bitcoin blockchain [44], since the blockchain will fork into m branches with m unique mining nodes. However, in this section, our goal is to merely demonstrate how $\mathcal{A}$ double-spends by exploiting the partitioning. To that end, we only present one attack construction in which $\mathcal{A}$ orchestrates the mining on two branches of the public chain and eventually releases the longest chain with a double-spent transaction.

In Fig. 8.12, we present an attack construction showing how $\mathcal{A}$ double-spends in the SyncAttack. We categorize the mining nodes in two groups, namely $M_i \in N_i$ and $M_e \in N_e$. After executing algorithm 9, $\mathcal{A}$ waits for ≈11 days until $|M_e|$=0 and all the mining nodes are in M_i[5]. $\mathcal{A}$ then estimates the hash rate of each mining node based on their block releasing frequency and further categorizes them into M_1 and M_2, with a combined hash rate of α and β, respectively. Next, $\mathcal{A}$ generates a transaction tx and conflicting transaction $tx^{'}$ using the same "Unspent Transaction Output" (UTXO). $\mathcal{A}$ relays tx to M_1 and a user A, and $tx^{'}$ to M_2 and another user B.

[5]The 11 days estimate for $|M_e|$=0 is based on our measurements in §8.3. Once all the mining nodes leave M_e, $\mathcal{A}$ can launch the double-spend attack.

Double-spending in the SyncAttack

Input: Mining nodes $M_i \in N_i$, $M_e \in N_e$, and adversary $\mathcal{A}$. Each node mines on C. Initially, M_i=0 and $M_e = \min x_i \in X$ (1).

Churn: $\mathcal{A}$ waits for 11 days until $M_i = \min x_i \in X$ and M_e=0 so that $\mathcal{A}$ has a complete control over all the mining nodes. During churn, whenever any $m_i \in M_i$ or $m_i \in M_e$ produces a block, $\mathcal{A}$ relays that block to all nodes N_r through A_r. As a result, all nodes have the same ledger $\mathcal{C}$ on their blockchain.

Hash Rate Estimation: Once $M_i = \min x_i \in X$, $\mathcal{A}$ measures h_i for each $m_i \in M_i$. The hash rate can be measured by annotating a mining node with its block mining frequency.

Split Miners: $\mathcal{A}$ then splits M_i into M_1 and M_2. $M_1 \leftarrow \alpha$ is the hash rate of M_1 and $M_2 \leftarrow \beta$ is the hash rate of M_2. Although, $\mathcal{A}$ can split M_i by any factor to obtain the desirable values for α and β, however, we assume that α=0.6 and β=0.4. The attack will also succeed for any other values for α and β.

Issue Double-spent Transactions: $\mathcal{A}$ selects two users A and B with non-mining nodes $n_a \in N_i$ and $n_b \in N_i$, respectively. $\mathcal{A}$ then generates a transaction tx a double-spent transaction $tx^{'}$ from the same UTXO [28]. For tx, $\mathcal{A}$ selects A is the recipient, and for $tx^{'}$, $\mathcal{A}$ selects B as the recipient. For each transaction, $\mathcal{A}$ sets a high mining fee and sends tx to M_1 and $tx^{'}$ to M_2.

Block Race: Assuming the b_r to be the latest block on $\mathcal{C}$, when the block race starts, the mining nodes in M_1 mine $b_{r+1} \preceq b_r$ before the mining nodes in M_2 mine $b^{'}_{r+1} \preceq b_r$ with 0.6 probability [18, 28]. The blockchain $\mathcal{C}$ splits into $\mathcal{C}_1 \preceq \mathcal{C}$ and $\mathcal{C}_2 \preceq \mathcal{C}$. The block b_{r+1} contains t_x and the block $b^{'}_{r+1}$ contains $tx^{'}$.

Block Release: Upon receiving b_{r+1} and $b^{'}_{r+1}$, $\mathcal{A}$ relays b_{r+1} to the mining nodes in M_1 and $b^{'}_{r+1}$ to the mining nodes in M_2. Additionally, $\mathcal{A}$ relays b_{r+1} to n_a and $b^{'}_{r+1}$ to n_b.

Receiving Product: Once $\mathcal{C}_1$ becomes k blocks long (typically $k = 6$ is the confirmation factor in Bitcoin [12]), A delivers the product to $\mathcal{A}$ or spends $tx^{'}$ with another user. Similarly, when $\mathcal{C}_2$ becomes $k = 6$ blocks long, B delivers the product to $\mathcal{A}$ or spends $tx^{'}$ with another user.

Dissolving Fork: Once $\mathcal{A}$ receives a product from both A and B, $\mathcal{A}$ releases the longer chain $\mathcal{C}_1$ to all the *reachable* nodes N_r in the Bitcoin network. Complying with the longest chain rule [16, 28], all mining and non-mining nodes switch to $\mathcal{C}_1$ and discard $\mathcal{C}_2$. $\mathcal{A}$ double-spends since $tx^{'}$ is invalidated.

Figure 8.12: Double-spending in the SyncAttack where $\mathcal{A}$ orchestrates mining on two blockchain branches and generates conflicting transactions on each branch. When $\mathcal{A}$ receives the reward for each transaction, $\mathcal{A}$ releases the longest branch to diffuse the fork. Note that despite diffusing the fork, $\mathcal{A}$ still controls N_i and can always re-launch the attack.

When any mining node in M_1 or M_2 releases a block, $\mathcal{A}$ relays that block to other miners in that group. Since the mining nodes in M_1 do not receive a block from the mining nodes in M_2, the blockchain $\mathcal{C}$ forks into $\mathcal{C}_1 \leftarrow M_1$ and $\mathcal{C}_2 \leftarrow M_2$. Moreover, $\mathcal{A}$ relays $\mathcal{C}_1$ to user A and $\mathcal{C}_2$ to user B. Eventually, when both $\mathcal{C}_1$ and $\mathcal{C}_2$ acquire a $k-$confirmation promised by $\mathcal{A}$ to both A and B, $\mathcal{A}$ receives products from both users and releases the longest branch to diffuse the fork. In Fig. 8.12, we assume that $\alpha > \beta$, thus $\mathcal{C}_1$ is longer than $\mathcal{C}_2$ and $tx^{'}$ is rejected. As a result, user B is tricked and $\mathcal{A}$ double-spends without using any mining power.

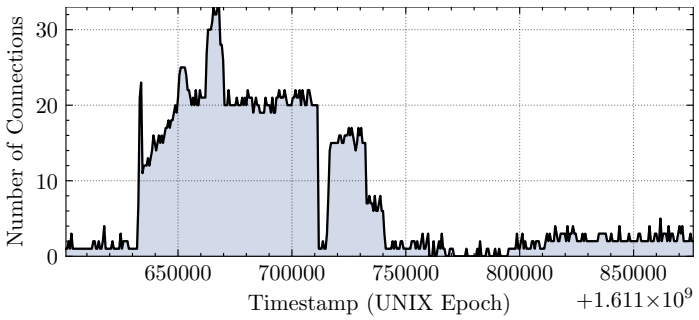


Figure 8.13: Number of incoming connections from the same IP address recorded on our *reachable* nodes. We observed instances where the node received up to 33 connections from the same IP address, indicating an attempt to target our node.

8.4.3 Ongoing Attacks

Considering the SyncAttack feasibility, it is logical to assume that malicious nodes could be exploiting the Bitcoin Core vulnerability to deteriorate the network synchronization or to perform other malicious activities. To investigate that and to identify those malicious nodes, we set up a *reachable* node and observe the number of incoming connections that use the same IP address. We conducted our experiment for three days and analyzed the IP addresses of the incoming connections using the RPC API at 10 minutes interval.

In Fig. 8.13, we report our results showing instances where our node received up to 33 incoming connections from the same IP address. Upon further inspecting the *debug.log* file, we found that each connection exhibited characteristics of $a_i \in A_u$ by continuously requesting the Bitcoin blocks from the *genesis* block. Fig. 8.13 also shows two time windows spanning ≈21 hours and ≈4.5 hours, in which the number of incoming connections significantly increased. Given that each Bitcoin node randomly selects an IP address for the outgoing connection, it is improbable that up to 33 *unreachable* nodes selected the same IP address to request the blockchain. Therefore, the two anomalies observed in Fig. 8.13 suggest a likely malicious activity. Although, the activities observed at our node do not represent all characteristics of the SyncAttack, they however clearly show that malicious nodes are exploiting vulnerabilities to affect the network synchronization.

8.4.4 SyncAttack Countermeasures

In this section, we present the SyncAttack countermeasures, some of which have been deployed and released in [2] while others are currently in development. For the SyncAttack, we propose application-specific and network-specific defenses that can be deployed in Bitcoin.

Application-specific Defenses. Our analysis in §8.4.1 shows that an adversary can occupy all the available connection slots of the *reachable* nodes by hosting only 20 Docker containers with unique IP addresses. This is made possible by the vulnerability in Bitcoin Core design, which allows one IP address to occupy more than 65K connection slots in the network. Therefore, a naïve approach to counter the attack is by removing the IP address and port concatenation, and only allowing 1 incoming connection per IP address. This policy will raise the attack cost to ≈4 Million USD since the adversary will then be required to acquire thousands of IP addresses [106]. We have deployed the *"1 IP address per incoming connection"* policy in the Bitcoin Core [2].

It can be argued that the *"1 IP address per incoming connection"* policy affects the *unreachable* nodes behind NAT. In other words, two *unreachable* nodes behind NAT cannot connect to the same *reachable* node. However, given that there are more than 11K *reachable* nodes, the current network can easily support a large volume of *unreachable* nodes behind each NAT. If each node establishes eight outgoing connections, then up to 1,375 *unreachable* nodes behind a NAT can be supported by the *reachable* nodes in the current network.

Network-specific Defenses. Network churn cannot be avoided due to the *permissionless* nature of the network. However, Bitcoin miners can deploy network-specific defense techniques to minimize the risk of partitioning created by churn. In §10, we observed that $\mathcal{A}$ completely controls the communication model in N_i, allowing $\mathcal{A}$ to violate the ideal functionality. Since all mining nodes experience churn and become part of N_i, they are vulnerable to the SyncAttack. In contrast, if the mining nodes resist churn and persistently stay in N_e, the risk of SyncAttack can be reduced. Due to the pre-attack stable outgoing connections R_o in N_e, $\mathcal{A}$ only partially controls the communication in N_e. Although the departure of a node from N_e reduces R_o by eight connections, we observe

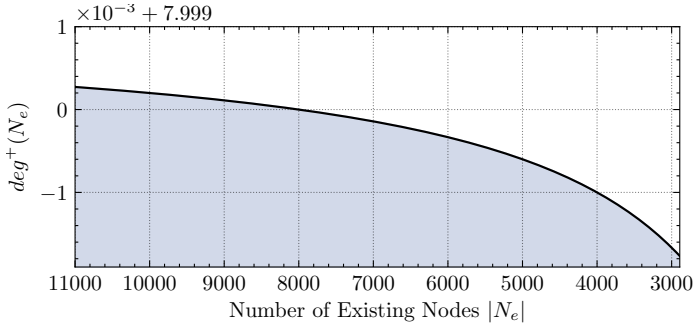


Figure 8.14: The change in $\deg^+(N_e)$ as $|N_e|$ decreases from 11K to 2,890.

that it does not significantly decrease $\deg^+(N_e)$, preventing $\mathcal{A}$ from completely controlling N_e.

To analyze the effect of churn on N_e, we start from the pre-attack situation where $|N_i|$=0 and $N_e = N_r$=11K. The adversary then executes algorithm 9 to maintain R_a=0 during churn. As a result, the size $|N_e|$ decreases with time along with the number of edges among the honest nodes. §8.3.2.1 provides us the minimum value for $|N_e|$ which is the total number of persistent nodes $|N_p|$=2,890 in the network. Using these values, we decrease $|N_e|$ from 11K to 2,890 and remove 8 edges with each node. With each node removal, we calculate $\deg^+(N_e)$ as follows.

$$\deg_(N_e) = \left\lceil \frac{|N_e|}{|N_e|-1} \log_{O_i}(|N_e|) \right\rceil \tag{8.5}$$

In Fig. 8.14, we plot $\deg^+(N_e)$ against $|N_e|$, showing only a marginal decrease in $\deg^+(N_e)$ with the maximum and minimum values of 8 and 7.997, respectively. From these results, we conclude that compared to N_i, N_e is relatively more secure against SyncAttack since $\mathcal{A}$ cannot occupy the stable outgoing connections of nodes in N_e. If mining nodes stay in N_e and maintain persistent outgoing connections with each other, they can resist the SyncAttack.

From our analysis in §8.4.2.1, we acknowledge that the network-specific defenses do not offer a complete protection against SyncAttack since the new mining nodes join the Bitcoin network. If R_a=0, those miners will become part of N_i and $\mathcal{A}$ can trigger a mining race between miners in N_i

and N_e. Acknowledging this possibility, we emphasize that both application-specific defenses and the network-specific defenses must be deployed to fully counter the SyncAttack.

8.5 Summary

In this chapter, we incorporate network synchronization in the Bitcoin security model and evaluate its robustness in the *permissionless* network. Our measurements and analysis present a contrast between the ideal functionality and the real world network behavior to expose various attack vectors that can be exploited to deteriorate the network synchronization and violate the Bitcoin blockchain *consistency* property. Especially new to the Bitcoin security model is our observation that churn can be exploited to partition the network and deteriorate the network synchronization. We formally analyze the churn-based partitioning by presenting SyncAttack that allows an adversary to double-spend without using any mining power. Moreover, we identify a vulnerability in Bitcoin Core that can be exploited to significantly lower the SyncAttack cost from $4 Million to only about $1,000. Realizing the feasibility of SyncAttack, and observing that malicious nodes are possibly exploiting the vulnerability to deteriorate network synchronization, we patch the vulnerability in Bitcoin Core and propose application-specific and network-specific defenses to counter the SyncAttack.

With the SyncAttack, we conclude our attack surface analysis in this dissertation by presenting the most feasible network layer attack that violates the Bitcoin blockchain *consistency*. Our work opens new directions in the security evaluation of *permissionless* blockchain systems by emphasizing the need to incorporate the realistic network synchronization in their security models.

CHAPTER 9: CONCLUSION

Blockchains have become a new paradigm in the distributed systems, enabling secure asset exchange among entities with competing interests. Blockchain systems are fast expanding, particularly in the form of cryptocurrencies such as Bitcoin and Ethereum that have a joint market capitalization of over ≈$850 Billion. Due to a high market capitalization, blockchain-based cryptocurrencies are often attacked by adversaries for monetary gains.

In this dissertation, we have comprehensively analyzed the blockchain attack surface. We observed that the blockchain attack surface can be broadly categorized into attacks related to (1) the application-specific policies in blockchain systems, (2) the cryptographic constructs of the blockchain data structure, and (3) the P2P network formed by the blockchain nodes. Among these three categorizes, the cryptographic constructs generally remain the same across most blockchain systems while the application-specific policies and the P2P network intricacies may vary significantly. As a result, the adversaries typically exploit the application-specific policies or the network layer irregularities to launch the attacks. Therefore, in this dissertation, we took a top-down approach in our attack surface analysis, starting with the application-specific attacks.

In the application-specific attacks, first we conducted the static and dynamic analysis of in-browser cryptojacking, an attack that involves hijacking resources of a machine to covertly mine cryptocurrency (Chapter 3). Using an *unsupervised* machine learning approach on the code-based features, we were able to uniquely distinguish the cryptojacking scripts from other forms of *JavaScript* codes with ≈96% accuracy. In the dynamic analysis, we evaluated the resource exploitation of a target device through cryptojacking, and found that cryptojacking consumes a significant processing power of the target device causing excessive battery drainage in the battery powered devices. Finally, by examining the limitations of the existing countermeasures, we proposed more robust countermeasures for cryptojacking using insights from our dynamic analysis.

In our second work on the application-specific attacks, we uncovered a distributed denial-of-service

attacks on PoW-based blockchain systems (Chapter 4). We observed a high correlation between the transaction fee and the size of the memory pool that stores unconfirmed transactions prior to mining. We observed that the memory pool size can be trivially increased through dust transactions thereby forcing benign users to pay a high transaction fee in order to prioritize their transactions. To counter the attack, we proposed fee-based an age-based countermeasures that removed malicious transactions from the memory pool and optimized the memory pool size.

Following our top-down approach, we then analyzed the network layer attacks on the blockchain systems, starting with the spatial, temporal, and spatio-temporal attacks on the Bitcoin network (Chapter 5). In our measurements-driven analysis, we observed (1) a biased distribution of Bitcoin nodes in the physical network, and (2) a weak network synchronization among those nodes. Based on those observations, we proposed spatial, temporal, and spatio-temporal partitioning attacks that allow a malicious miner to fork the blockchain with less than 51% of the network hash rate.

In Chapter (6), we expanded on the temporal partitioning attack by conducting a root cause analysis for weak synchronization in the Bitcoin network. We conducted large-scale measurements and characterized the impact of (1) *unreachable* nodes, (2) network addressing protocol, (3) block relaying protocol, and (4) network churn on synchronization. Our results revealed that among all those factors, churn had the most significant impact on network synchronization. A high churn among the synchronized nodes led to variations in the network outdegree which then deteriorated network synchronization. Additionally, we also observed that (1) the real world Bitcoin network is asynchronous and blocks reach miners at different times, and (2) network synchronization has not been formally incorporated in the Bitcoin security model.

In Chapter (7), we combined the application-specific policies (*i.e.* block mining strategies) with the network irregularities (*i.e.* asynchronous network) to present the *HashSplit* attack that allows a miner to violate the blockchain consistency and chain quality with as low as 26% hash rate. We proposed an ideal functionality that embraced the mining centrality in the Bitcoin network and characterized the behavior of mining and non-mining nodes. We then conducted measurements to

detect the mining nodes and observed asynchronous block propagation among them. Using that knowledge, we constructed the *HashSplit* attack that allows an adversary to orchestrate concurrent mining on two branches of the public chain to violate the blockchain consistency and chain quality with a high probability. We also proposed and implemented the *HashSplit* countermeasures.

The last chapter in the dissertation (Chapter 8) concludes our attack surface analysis by presenting the most cost effective attack on the Bitcoin P2P network. Towards that, we first formulated the ideal functionality to incorporate network synchronization in the Bitcoin security model. We then discovered that the network synchronization can be violated by exploiting the network churn. Using that knowledge, we presented SyncAttack, an attack that allows an adversary to occupy all the connection slots of the *reachable* nodes and create a partitioning between the existing nodes and the incoming nodes. Moreover, we discovered a vulnerability in Bitcoin Core that reduces the attack cost from $4 Million to only about $1000. Compared to any network layer attack presented in this work or in the literature, SyncAttack is the most feasible partitioning attack on the Bitcoin network. We conclude Chapter 8 by deploying the SyncAttack countermeasures in Bitcoin Core.

In conclusion, this dissertation makes foundational contributions to the distributed systems security by uncovering various novel attacks related to the application-specific policies and the network layer intricacies of the blockchain systems. Our theoretical modeling, measurements, attacks, and countermeasures open new directions in the security evaluation of blockchain systems.

APPENDIX: COPYRIGHT INFORMATION

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

Dine and Dash: Static, Dynamic, and Economic Analysis of In-Browser Cryptojacking
Muhammad Saad, Aminollah Khormali, Aziz Mohaisen
2019 APWG Symposium on Electronic Crime Research (eCrime)

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Muhammad Saad	20-11-2019
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the

IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

Questions about the submission of the form or manuscript must be sent to the publication's editor.
Please direct all questions about IEEE copyright policy to:
IEEE Intellectual Property Rights Office, copyrights@ieee.org, +1-732-562-3966

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

Mempool Optimization for Defending Against DDoS Attacks in PoW-based Blockchain Systems
Muhammad Saad and Laurent Njilla and Charles Kamhoua and Joongheon Kim and DaeHun Nyang and Aziz Mohaisen
2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Muhammad Saad	11-03-2019
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the

IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

Questions about the submission of the form or manuscript must be sent to the publication's editor.
Please direct all questions about IEEE copyright policy to:
IEEE Intellectual Property Rights Office, copyrights@ieee.org, +1-732-562-3966

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

Partitioning Attacks on Bitcoin: Colliding Space, Time, and Logic
Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, Aziz Mohaisen
2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Muhammad Saad	15-04-2019
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the

IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

Questions about the submission of the form or manuscript must be sent to the publication's editor.
Please direct all questions about IEEE copyright policy to:
IEEE Intellectual Property Rights Office, copyrights@ieee.org, +1-732-562-3966

LIST OF REFERENCES

[1] A. Ahmad, M. Saad, and A. Mohaisen. Secure and transparent audit logs with *BlockAudit*. *J. Netw. Comput. Appl.*, 145, 2019. https://doi.org/10.1016/j.jnca.2019.102406.

[2] Anonymous. Bitcoin synchronization attack. https://anonymous.4open.science/r/106b2297-8daf-4b75-a209-6468a8dc91c1/.

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

[4] Antpool. Antpool stratum address, 2018. https://www.antpool.com/.

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

[6] B. Badge. Plato tool, 2016. https://github.com/es-analysis/plato.

[7] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver. Stressing out: Bitcoin "stress testing". In *Financial Cryptography and Data Security*, pages 3–18, 2016. https://doi.org/10.1007/978-3-662-53357-4_1.

[8] M. Bastiaan. Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In *Technical Report*, 2015. https://goo.gl/cNACCq.

[9] J. C. Bezdek, R. Ehrlich, and W. Full. Fcm: The fuzzy c-means clustering algorithm. *Computers and Geosciences*, 10:191–203, 1984. https://bit.ly/3tIQgNF.

[10] Blockchain. Hashrate distribution, 2018. https://blockchain.info/pools.

[11] A. Bruns, A. Kornstädt, and D. Wichmann. Web application tests with selenium. *IEEE Softw.*, 26(5):88–91, 2009. https://doi.org/10.1109/MS.2009.144.

[12] BTC. Btc.com stratum address, 2018. https://bit.ly/2N9NQH6.

[13] Coinhive. Monero JavaScript Mining, 2018. https://coinhive.com/documentation.

[14] B. Community. Antpool hash rate. https://www.bitcoinmining.com/images/bitcoin-mining-pool-hash-rate-distribution.png.

[15] B. Community. Fork monitor. https://forkmonitor.info/stale/btc/666833.

[16] B. Community. Global cryptocurrency market charts — coinmarketcap. https://coinmarketcap.com/charts/.

[17] B. Community. Bitcoin Data from Blockchain.info, 2017. https://blockchain.info/charts/transaction-fees-usd.

[18] B. Community. Bitcoin core, 2018. https://github.com/bitcoin/bitcoin.

[19] B. Community. Bitnodes: Global bitcoin nodes distribution, 2018. https://bitnodes.earn.com/.

[20] B. Community. Developer's Guide, Confirmation Score, Transaction Fee and Miner Fee, Minimum Relay Fee, UTXO, Memory Pool, Child Pays for Parent, Raw Transactions, 2018. https://bitcoin.org/en/developer-reference#rpc-quick-reference.

[21] B. Community. Stratum mining protocol, 2018. https://en.bitcoin.it/wiki/Stratum_mining_protocol.

[22] B. Community. Antminer, 2019. https://m.bitmain.com/.

[23] B. Community. Bitnodes: Discovering all reachable nodes in bitcoin, 2019. https://bitnodes.earn.com/.

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

[25] B. Community. Bitcoin forks and orphaned blocks chart, 2020. https://www.blockchain.com/charts/n-orphaned-blocks.

[26] E. Community. Earn: Earn money by answering messages and completing tasks, 2018. https://earn.com.

[27] J. Community. JSECoin: Digital currency - designed for the web, 2018. https://jsecoin.com/.

[28] M. Community. Monero cryptocurrency, 2018. https://monero.org/.

[29] S. Community. Selenium browser automation, 2018. https://www.seleniumhq.org/docs/.

[30] M. Corallo. Bitcoin improvement proposal 152. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[31] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125, 2016. https://doi.org/10.1007/978-3-662-53357-4_8.

[32] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert. ZOZZLE: fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, 2011. http://static.usenix.org/events/sec11/tech/full_papers/Curtsinger.pdf.

[33] C. Decker. Bitcoin dns seeders. https://bitcoinstats.com/network/dns-servers/.

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

[35] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions. In *International Conference on Financial Cryptography and Data Security*, volume 11598, pages 550–566, 2019. https://doi.org/10.1007/978-3-030-32101-7_32.

[36] J. Donier and J.-P. Bouchaud. Why do markets crash? bitcoin data offers unprecedented insights. *PloS one*, 10:e0139356, 03 2015. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0139356.

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

[38] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark. A first look at browser-based cryptojacking. In *European Symposium on Security and Privacy Workshops*, pages 58–66, 2018. https://doi.org/10.1109/EuroSPW.2018.00014.

[39] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454, 2014. https://doi.org/10.1007/978-3-662-45472-5_28.

[40] F2Pool. F2pool stratum address, 2018. https://www.f2pool.com/help.

[41] M. Fadhil, G. Owenson, and M. Adda. Locality based approach to improve propagation delay on the bitcoin peer-to-peer network. In *Symposium on Integrated Network and Service Management*, 2017. https://doi.org/10.23919/INM.2017.7987328.

[42] A. Feder, N. Gandal, J. T. Hamrick, and T. Moore. The impact of DDoS and other security shocks on bitcoin currency exchanges: evidence from mt. gox. *J Cyber Secur*, 3(2):137–144, 2017. https://doi.org/10.1093/cybsec/tyx012.

[43] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on software engineering*, 25(5):675–689, 1999. https://doi.org/10.1109/32.815326.

[44] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *International Cryptology Conference on Advances in Cryptology*, pages 291–323, 2017. https://doi.org/10.1007/978-3-319-63688-7_10.

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

[46] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In C. N. P. Jr., A. Hahn, K. R. B. Butler, and M. Sherr, editors, *Computer Security Applications Conference*, pages 326–335, 2014. https://doi.org/10.1145/2664243.2664267.

[47] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Conference on Computer and Communications Security*, pages 3–16, 2016. https://doi.org/10.1145/2976749.2978341.

[48] S. Goldberg. Why is it taking so long to secure internet routing? *Commun. ACM*, 57(10):56–63, Sept. 2014. http://doi.acm.org/10.1145/2659899.

[49] S. Goldberg and E. Heilman. Technical perspective: The rewards of selfish mining. *Commun. ACM*, 61(7):94, 2018. https://doi.org/10.1145/3213006.

[50] A. Greenberg. Hacker redirects traffic from 19 internet providers to steal bitcoins, Jun 2017. https://www.wired.com/2014/08/isp-bitcoin-theft/.

[51] I. M. Group. IP address marketplace: Worldwide, Jan 2021. https://ipv4marketgroup.com/ipv4-pricing/.

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

[53] C. Grunspan and R. Pérez-Marco. On profitability of selfish mining. *CoRR*, abs/1805.08281, 2018. http://arxiv.org/abs/1805.08281.

[54] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017. https://bit.ly/3jBOjOg.

[55] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[56] G. Hong, Z. Yang, S. Yang, L. Zhang, Y. Nan, Z. Zhang, M. Yang, Y. Zhang, Z. Qian, and H. Duan. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Conference on Computer and Communications Security*, pages 1701–1713, 2018. https://doi.org/10.1145/3243734.3243840.

[57] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis. Churn in the bitcoin network: Characterization and impact. In *International Conference on Blockchain and Cryptocurrency*, pages 431–439, 2019. https://doi.org/10.1109/BLOC.2019.8751297.

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

[59] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore. Game-theoretic analysis of DDoS attacks against bitcoin mining pools. In *Financial Cryptography and Data Security*, pages 72–86, 2014. https://doi.org/10.1007/978-3-662-44774-1_6.

[60] L. D. Jr. Bitcoin seed file. http://luke.dashjr.org/programs/bitcoin/files/charts/seeds.txt, 2020.

[61] R. Keramidas. No coin web extension to detect cryptojacking, Feb 2018. https://github.com/keraf/NoCoin.

[62] A. Kharraz, Z. Ma, P. Murley, C. Lever, J. Mason, A. Miller, N. Borisov, M. Antonakakis, and M. Bailey. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In *The World Wide Web Conference*, pages 840–852, 2019. https://doi.org/10.1145/3308558.3313665.

[63] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security Symposium*, pages 279–296, 2016. https://bit.ly/3rQrOZd.

[64] K. Kononova and A. Dek. Bitcoin carbon footprint: Mining pools based estimate methodology. In *International Conference on Information and Communication Technologies in Agriculture, Food and Environment*, pages 265–273, 2020. http://ceur-ws.org/Vol-2761/HAICTA_2020_paper39.pdf.

[65] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Conference on Computer and Communications Security*, 2018. http://doi.acm.org/10.1145/3243734.3243858.

[66] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of bitcoin mining. In *Proceedings of WEIS*, 2013. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2531518.

[67] Y. Kwon, D. Kim, Y. Son, E. Y. Vasserman, and Y. Kim. Be selfish and avoid dilemmas: Fork after withholding (FAW) attacks on bitcoin. In *Conference on Computer and Communications Security*, pages 195–209, 2017. https://doi.org/10.1145/3133956.3134019.

[68] S. Lee, S. Shin, and B. Roh. Abnormal behavior-based detection of shodan and censys-like scanning. In *International Conference on Ubiquitous and Future Networks*, pages 1048–1052, 2017. https://doi.org/10.1109/ICUFN.2017.7993960.

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

[70] T. Loechner. Pixalate unveils the list of sites secretly mining cryptocurrency, 2017.

[71] S. B. Mariem, P. Casas, M. Romiti, B. Donnet, R. Stütz, and B. Haslhofer. All that glitters is not bitcoin - unveiling the centralized nature of the BTC (IP) network. In *IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2020. https://doi.org/10.1109/NOMS47738.2020.9110354.

[72] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun. BITE: bitcoin lightweight client privacy using trusted execution. In *USENIX Security Symposium*, pages 783–800, 2019. https://bit.ly/2Z2QHo0.

[73] F. Memoria. 700 million stuck in 115,000 unconfirmed bitcoin transactions, 2017. https://goo.gl/mYX14V.

[74] S. G. Motlagh, J. V. Misic, and V. B. Misic. Modeling of churn process in bitcoin network. In *International Conference on Computing, Networking and Communications*, pages 686–691, 2020. https://doi.org/10.1109/ICNC47757.2020.9049704.

[75] R. Nagayama, R. Banno, and K. Shudo. Identifying impacts of protocol and internet development on the bitcoin network. In *Symposium on Computers and Communications*, pages 1–6. IEEE, 2020. https://doi.org/10.1109/ISCC50000.2020.9219639.

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

[77] C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *International Conference on Dependable Systems and Networks*, pages 579–590, 2017. https://doi.org/10.1109/DSN.2017.44.

[78] T. Neudecker, P. Andelfinger, and H. Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In *International Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing*, pages 358–367, 2016. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070.

[79] D. Ocean. Spin up your virtual machine in just 55 seconds. https://try.digitalocean.com/.

[80] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016. https://bit.ly/3p1GRNP.

[81] M. Polasik, A. I. Piotrowska, T. P. Wisniewski, R. Kotkowski, and G. Lightfoot. Price fluctuations and the use of bitcoin: An empirical inquiry. *International Journal of Electronic Commerce*, 20(1):9–49, Sept. 2015. https://bit.ly/2YYqBT0.

[82] M. Rahouti, K. Xiong, and N. Ghani. Bitcoin concepts, threats, and machine-learning security solutions. *IEEE Access*, 6:67189–67205, 2018. https://bit.ly/2Z05K1w.

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

[84] B. Reward. Bitcoin block reward halving countdown, 2018. http://www.bitcoinblockhalf.com/.

[85] RIR. Autonomous systems in the world, 2018. https://tinyurl.com/yaz73jnb.

[86] A. Robachevsky. 14,000 incidents: A 2017 routing security year in review, Jan 2018. https://goo.gl/MtiVus.

[87] M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011. http://arxiv.org/abs/1112.4980.

[88] M. Rosenfeld. Analysis of hashrate-based double spending. *CoRR*, abs/1402.2009, 2014. http://arxiv.org/abs/1402.2009.

[89] T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2P mixing and unlinkable bitcoin transactions. In *Network and Distributed System Security Symposium*, 2017. https://bit.ly/2MIvgWU.

[90] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld. Digging into browser-based crypto mining. In *ACM Internet Measurement Conference*, New York, USA, 2018. http://doi.acm.org/10.1145/3278532.3278539.

[91] M. Saad, A. Anwar, A. Ahmad, H. Alasmary, M. Yuksel, and A. Mohaisen. Routechain: Towards blockchain-based secure and efficient BGP routing. In *International Conference on Blockchain and Cryptocurrency*, pages 210–218, 2019. https://doi.org/10.1109/BLOC.2019.8751229.

[92] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen. Partitioning attacks on bitcoin: Colliding space, time, and logic. In *International Conference on Distributed Computing Systems*, pages 1175–1187, 2019. https://bit.ly/3aa0sat.

[93] M. Saad, A. Khormali, and A. Mohaisen. Dine and dash: Static, dynamic, and economic analysis of in-browser cryptojacking. In *APWG Symposium on Electronic Crime Research*, pages 1–12, 2019. https://doi.org/10.1109/eCrime47957.2019.9037576.

[94] M. Saad and A. Mohaisen. Towards characterizing blockchain-based cryptocurrencies for highly-accurate predictions. In *IEEE Conference on Computer Communications*, pages 704–709, 2018. https://doi.org/10.1109/INFCOMW.2018.8406859.

[95] M. Saad, L. Njilla, C. A. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen. Mempool optimization for defending against ddos attacks in pow-based blockchain systems. In *International Conference on Blockchain and Cryptocurrency*, pages 285–292, 2019. https://doi.org/10.1109/BLOC.2019.8751476.

[96] M. Saad, L. Njilla, C. A. Kamhoua, and A. Mohaisen. Countering selfish mining in blockchains. In *International Conference on Computing, Networking and Communications*, pages 360–364, 2019. https://doi.org/10.1109/ICCNC.2019.8685577.

[97] M. Saad, J. Spaulding, L. Njilla, C. A. Kamhoua, S. Shetty, D. Nyang, and D. A. Mohaisen. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Commun. Surv. Tutorials*, 22(3):1977–2008, 2020. https://doi.org/10.1109/COMST.2020.2975999.

[98] M. F. Sallal, G. Owenson, and M. Adda. Proximity awareness approach to enhance propagation delay on the bitcoin peer-to-peer network. In *International Conference on Distributed Computing Systems*, pages 2411–2416, 2017. https://bit.ly/3qb60a4.

[99] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532, 2016. https://doi.org/10.1007/978-3-662-54970-4_30.

[100] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in Probability and Statistics. Wiley, 1992. https://bit.ly/3rDlIey.

[101] Y. Shahsavari, K. Zhang, and C. Talhi. Performance modeling and analysis of the bitcoin inventory protocol. In *International Conference on Decentralized Applications and Infrastructures*, pages 79–88, 2019. https://bit.ly/3b5bop5.

[102] SLM. In-browser cryptojacking: What is it and how can you avoid it?, 2018. https://supremelevelmedia.com/browser-cryptojacking-can-avoid/.

[103] S. Solat and M. Potop-Butucaru. Zeroblock: Preventing selfish mining in bitcoin. *arXiv preprint arXiv:1605.02435*, 2016. http://arxiv.org/abs/1605.02435.

[104] C. B. Staff. 21 top examples of javascript, 2017. https://tinyurl.com/y8wqarpb.

[105] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. A. Gunter, F. Zaffar, M. Caesar, and N. Borisov. Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *International Symposium on Research in Attacks, Intrusions and Defenses*, pages 287–310, 2017. https://doi.org/10.1007/978-3-319-66332-6_13.

[106] 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 *IEEE Symposium on Security and Privacy*, pages 894–909, 2020. https://doi.org/10.1109/SP40000.2020.00027.

[107] M. Vasek, M. Thornton, and T. Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *Financial Cryptography and Data Security*, pages 57–71. Springer, 2014. https://doi.org/10.1007/978-3-662-44774-1_5.

[108] C. Wang, X. Chu, and Y. Qin. Measurement and analysis of the bitcoin networks: A view from mining pools. In *International Conference on Big Data Computing and Communications*, pages 180–188, 2020. https://bit.ly/3p6dsBV.

[109] P. Wang, H. Deng, Y. M. Wang, Y. Liu, and Y. Zhang. Kernel density estimation based gaussian and non-gaussian random vibration data induction for high-speed train equipment. *IEEE Access*, 8:90914–90923, 2020. https://bit.ly/2OnIlFr.

[110] A. H. Watson, D. R. Wallace, and T. J. McCabe. *Structured testing: A testing methodology using the cyclomatic complexity metric*, volume 500. US Department of Commerce, Technology Administration, 1996.

[111] Wizsche. Malicious javascript dataset. `https://github.com/geeksonsecurity/js-malicious-dataset.git`, 2017.

[112] X. Yang. List of top Alexa websites with web-mining code embedded on their homepage, 2017. `https://tinyurl.com/ybo6u4pf`.

[113] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *ACM Internet Measurement Conference*, pages 373–380, 2014. `https://tinyurl.com/ybqmcjmb`.

[114] Z. Zhang, Y. Zhang, Y. C. Hu, and Z. M. Mao. Practical defenses against BGP prefix hijacking. In *Conference on Emerging Network Experiment and Technology*, page 3, 2007. `http://doi.acm.org/10.1145/1364654.1364658`.