SIMULATION-BASED EVALUATION OF TRANSACTION FINALITY IN BITCOIN USING CNSIM

AMIRREZA RADJOU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2025

Abstract

Blockchain technologies, such as Bitcoin, have long been regarded as holding the potential to fundamentally transform the way societies conduct economic transactions. Thanks to their decentralized nature, blockchain networks are able to process such transactions without the need to trust centralized and socially validated institutions, which are exposed to a variety of single-point-of-failure threats. Rather, a network of interconnected nodes, i.e., a consensus network, handles transaction validation through following a consensus protocol. A great many such protocols have been introduced, often based on very different design philosophies. Prior to any of them being trusted for real world use, such consensus protocols need to be thoroughly evaluated with respect to their security, performance, and resilience.

Given that the scale of such networks is meant to be substantial, experimental evaluation via their reproduction in a lab is challenging. Rather, simulations of abstract models of such consensus networks allows practical, cost-effective reproduction of their operation while focusing on evaluating properties and phenomena of interest. Thus, several blockchain simulators have been introduced in the literature for various purposes and objectives. However, despite the wealth of simulator software prototypes, there appears to be a need for a more general simulation framework, that is, a system of tools and processes for not only performing simulations but also for translating the simulation data into useful metrics. CNSim, a simulator developed at York University, is aimed at filling this gap by introducing a finality-based approach to conducting simulation studies, whereby the likelihood of transactions remaining irreversible within a finite horizon is the foundational construct for reproducibly and commensurably evaluating consensus networks in different contexts.

In this thesis, we adopt CNSim and its finality-based analysis approach and apply it to the Bitcoin blockchain in order to both study Bitcoin and evaluate the feasibility of the CNSim framework. To do so, I enhance CNSim by designing and implementing a framework to model adversarial behaviors, a capability it previously lacked. Specifically, I implement the Majority Attack to create a detailed simulation scenario for double-spending. Using this extended simulator, I conduct a systematic evaluation of the attack's impact on transaction finality, quantifying how network resilience degrades as malicious hash power increases. The findings provide valuable insights into the practical security limitations of the Bitcoin protocol and demonstrate the utility of a finality-based methodology for analyzing blockchain consensus mechanisms.

Acknowledgements

I would like to express my sincere gratitude for the support from the Dependable Internet of Things Applications (DITA) program, established by NSERC at York University. I am also deeply thankful to my supervisor, Professor Sotirios Liaskos, for his expert guidance and assistance throughout this journey. I would also like to extend my thanks to my examination committee, Manar Jammal and Zisis Poulos, for their valuable feedback. Finally, I would like to thank my friends and family for their constant belief in me and their endless patience and support. Most of all, I want to thank my partner for her unwavering love and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Blockchain technologies have long been recognized as carrying the promise of revolutionizing the way societies conduct economic transactions. Instead of relying on socially validated central authorities to guarantee the validity of transactions, blockchains employ a distributed network of anonymous nodes that collectively validate and record activity through a commonly adopted consensus algorithm. This decentralized structure reduces dependence on trusted intermediaries, enhances transparency, and strengthens resistance to censorship or tampering.

Bitcoin, introduced by Satoshi Nakamoto in 2008 [1], represents the first decentralized blockchain implementation enabling peer-to-peer digital transactions without centralized intermediaries. At its core, the Bitcoin network relies on a Proof-of-Work (PoW) consensus algorithm where network participants (miners) compete to solve complex cryptographic puzzles to validate transactions and extend the blockchain.

Transaction finality in Bitcoin emerges from the computational difficulty of altering previously confirmed blocks. As new blocks are appended, the probability of reversing transactions becomes exponentially smaller, creating a robust mechanism for ensuring ledger immutability.

However, recent research has revealed significant vulnerabilities in this mechanism, challenging previous assumptions about Bitcoin's consensus security. Saad et al. [2], for example, revealed that real-world Bitcoin networks operate under asynchronous conditions, contradicting fundamental synchrony assumptions critical to blockchain safety. Additionally, researchers such as Zhang and Preneel [3] have shown that no existing Proof-of-Work protocol achieves ideal chain quality or comprehensive attack resistance.

These findings underscore the need for robust evaluation of Bitcoin's consensus properties across diverse attack vectors and network impairments. This need extends to various other consensus protocols that have since emerged, including Proof-of-Stake [4], and different supporting data structures, such as DAGs [5]. However, conducting empirical evaluations on blockchain protocols is a non-trivial endeavor, due to both the scale of realistic blockchain networks, which is practically infeasible to reproduce in, e.g., a lab, and the heterogeneity of protocols and data structures, which makes comparisons difficult.

1.1 Finality and Consensus Quality

In evaluating blockchains, it is necessary to first identify the focal point of such evaluation. A critical aspect of blockchain's promise is *transaction finality* - the guarantee that once a transaction is recorded, it becomes irreversible. Once a transaction is deemed to be irreversible, the corresponding block in the business process that includes the transaction can be lifted. For example, a seller of a product will be comfortable to ship the product once the payment has been confirmed. In traditional centralized transaction processing systems, transactions,

once confirmed, often after a complex choreography of message exchanges among actors, as in the credit card system [6], they are deemed to be final. In blockchains, decentralization demands that a distributed consensus process must follow transaction arrival. However, depending on the design of the consensus process, the size of the network, its popularity, and many other parameters that are specific to the protocol, it is not always obvious if and when each transaction can be accepted as irreversible. Thus, the practical achievement of true transaction finality remains a complex challenge that demands rigorous scientific investigation.

This added complication that the decentralization requirement brings has been described through the *blockchain trilemma* [7], which posits that a blockchain system must balance three core properties: decentralization, security, and scalability. Achieving all three simultaneously is notoriously difficult, and researchers rely on systematic approaches to explore how design choices affect these dimensions.

Of interest here is the trade-off between decentralization and security. One of the ways by which finality can be compromised is through sophisticated attack vectors. Existing research has revealed vulnerabilities that threaten the core principles of decentralized consensus. Bonneau et al. [8] introduced the concept of bribery attacks, demonstrating how malicious actors could strategically purchase mining power to execute double-spending schemes. Liao and Katz [9] illustrated how minority attackers could incentivize miner collusion through large transaction fees, making double-spend attacks economically viable.

Network characteristics also play a crucial role in determining consensus quality. Mišić

et al. [10] demonstrated that block propagation latency, primarily influenced by round-trip network times, directly impacts fork occurrence frequency. Shahsavari et al. [11] further revealed how factors like network bandwidth, block size, and node connectivity modulate fork probabilities.

1.2 Simulation-based Evaluation

The above challenges require effective ways for evaluating the quality of consensus protocols. Simulation-based evaluation has been shown to be an important tool for performing such evaluations. While numerous blockchain simulators exist, each with its own strengths and limitations, there remains a need for frameworks that enable commensurable and systematic measurement of consensus network finality. CNSim, developed at York University, addresses this need by offering a theory of quality measurement of blockchain consensus networks along with a toolset to perform the measurements. The CNSim evaluation theory is based on an abstract characterization of finality of transactions, operationalized appropriately for each protocol to be analyzed, allowing comparability of analyses. The accompanying toolset is designed to investigate transaction finality under diverse network conditions and enables the systematic exploration of complex scenarios, including sophisticated attack models such as the Majority attack. By measuring critical parameters like throughput, latency, and security, CNSim allows understanding of the fundamental challenges to blockchain consensus mechanisms.

1.3 Research Questions

This research leverages and showcases CNSim's capabilities through an in-depth exploration of transaction finality in the Bitcoin network under specific attack and connectivity impairment scenarios. In particular, we focus on finality-centered metrics that can elucidate the resilience of Bitcoin's PoW consensus. The specific research questions we pose are:

- **RQ1:** Can CNSim effectively simulate Bitcoin and perform finality-based evaluation of its consensus mechanism?
- **RQ2:** How does Bitcoin's consensus quality vary under different attack intensities, and how do these findings compare with existing literature (e.g., [2, 8, 9])? In the concluding analysis (Chapter 4), we explicitly compare our simulation results with prior works.
- **RQ3:** What impact do various network characteristics and protocol configurations have on Bitcoin's consensus quality, and how do these results align with or diverge from current research findings (e.g., [10, 11, 12])? We discuss these outcomes in detail in Chapter 4, highlighting any alignment or discrepancies relative to the literature.
- **RQ4:** Are the finality-based approach to conducing blockchain simulation studies, as well as the associated metrics and visualizations useful for analyzing a consensus protocol such as bitcoin?

1.4 Contributions

This thesis makes the following contributions:

- **Adversarial Modeling Framework for CNSim:** We enhanced the CNSim simulator by designing and implementing a flexible framework to model malicious node behaviors, a capability it previously lacked. Using this new framework, we developed a detailed simulation of the Majority Attack [13, 14] to serve as a basis for analyzing its effects on Bitcoin's transaction finality.

- **Simulation-Based Evaluation of Finality:** We performed a comprehensive evaluation of Bitcoin's consensus mechanism under various Majority Attack intensities, network conditions, and protocol configurations. This analysis provides quantitative insights into the thresholds at which network security degrades and identifies key factors impacting transaction reliability.

- **Evaluation of the Finality-Based Approach:** Using the Bitcoin attack analysis as a case study, we evaluated the effectiveness of CNSim's finality-based methodology. We demonstrated that its metrics and visualizations, such as the consensus progression curve and transaction confidence score, are highly effective for conducting meaningful and comparative analyses of consensus protocols.

1.5 Thesis Organization

This thesis is organized as follows:

- **Chapter 2: Background** presents an overview of blockchain technology, including the components of the Bitcoin blockchain, various network attacks on the Bitcoin network, an examination of blockchain simulators, and a finality-based evaluation framework for blockchain networks.

- **Chapter 3: Finality-Based Blockchain Protocol Evaluation** outlines the design and implementation of the CNSim simulator and describes the enhancements made to simulate Bitcoin's transaction finality and security.

- **Chapter 4: Experimental Study** provides a detailed analysis of the simulation results, including visual representations of transaction finality, discussion of the impact of malicious behaviors on network stability, and comparisons with existing research.

- **Chapter 5: Conclusions and Future Work** summarizes the contributions and key findings of this research, discusses the implications of the finality-based methodology, and outlines directions for future work.

Chapter 2

Background

2.1 Blockchains and Consensus Networks

Blockchain technology has evolved into a wide-ranging paradigm for securing and decentralizing digital interactions. Although it was originally popularized by Bitcoin, the blockchain ecosystem now encompasses thousands of cryptocurrencies and an expanding variety of use cases, including supply chain management, digital identity systems, decentralized finance (DeFi), and healthcare data integrity [15, 16]. Despite the emergence of numerous blockchain platforms with alternative designs, Bitcoin remains the largest by market capitalization and the most historically tested. Its continued prominence motivates in-depth studies of Bitcoin's security assumptions, design trade-offs, and performance characteristics [17, 18].

In this chapter, we provide an overview of blockchain networks and the role of consensus and finality, focusing on the Bitcoin network, which is the focus of this thesis.

2.1.1 Consensus Protocols and Transaction Finality

Blockchain networks, such as Bitcoin, rely on a decentralized set of nodes to verify transactions through a *consensus protocol* [1]. These protocols differ in the techniques used to validate transactions and secure the network, ranging from computationally intensive Proof of Work (PoW) [19, 20] to resource-staking Proof of Stake (PoS) [21, 22, 23, 24] and other specialized variants. Yet, their overarching goal is consistent: ensure that a transaction is accepted as valid and becomes difficult or practically infeasible to reverse within a reasonable time frame. This property, commonly referred to as *transaction finality*, is crucial because it determines the point at which a transaction can be trusted to be permanent and immutable.

The ability to quickly and confidently declare a transaction final has profound implications for both the performance and the security of a blockchain system. From a user perspective, a transaction that achieves strong finality can be treated like an irreversible payment, reducing the risks associated with delayed settlement or potential rollbacks. This importance holds across different consensus mechanisms, highlighting that while implementations may vary, the underlying need for a reliable notion of finality remains a universal priority among decentralized ledgers.

We will next turn our focus to Bitcoin and explore the notions of consensus and finality there in more detail.

2.2 Bitcoin Blockchain

2.2.1 Overview

The **Bitcoin blockchain**, introduced by Satoshi Nakamoto in 2008 [1], fundamentally redefined digital value exchange by enabling a decentralized transaction platform independent of centralized authorities. Widely regarded as the pioneer of blockchain systems, Bitcoin provides a foundational context for understanding classical blockchain architectures, their key components, and the lifecycle of transactions within such networks [17, 18]. Despite the proliferation of alternative platforms, Bitcoin remains highly relevant due to its extensive network hash rate, which is a measure of mining activity and widespread investment in the health of the network, as well as its market capitalization, which shows the societal interest in the specific network. Its structure and operation demonstrate many principles – such as consensus-driven immutability and cryptographically secured transactions that inform our broader understanding of blockchain technology.

2.2.2 Architectural Components of the Bitcoin Network

Peer-to-Peer (P2P) Network

The Bitcoin network is organized as a decentralized **peer-to-peer (P2P)** system, wherein nodes directly exchange information without intermediaries (see Figure 2.1). Transactions, digitally signed by their originators, propagate through this network. Its decentralization ensures that no single point of failure compromises the system's integrity, thereby promoting

robustness against both faults and adversarial activities [25].

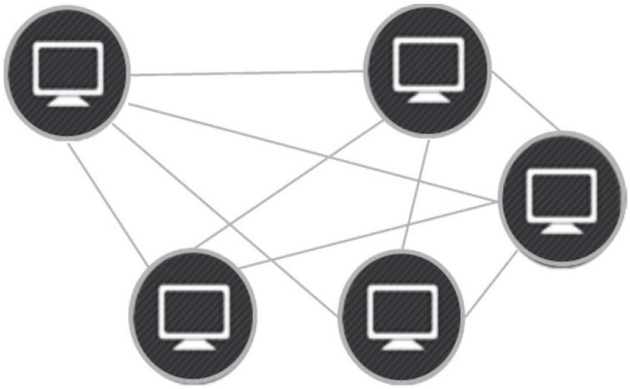

Figure 2.1: A conceptual diagram of a P2P network structure.

Blockchain

A **blockchain** is a public, distributed ledger that stores transactions in a verifiable and tamper-evident manner. Each block references its immediate predecessor via a cryptographic hash, forming an immutable chain (see Figure 2.2). This design ensures transparency and historical accuracy, as altering any single block would necessitate recomputing all subsequent blocks, an endeavor rendered prohibitively expensive by the underlying consensus mechanism [19].

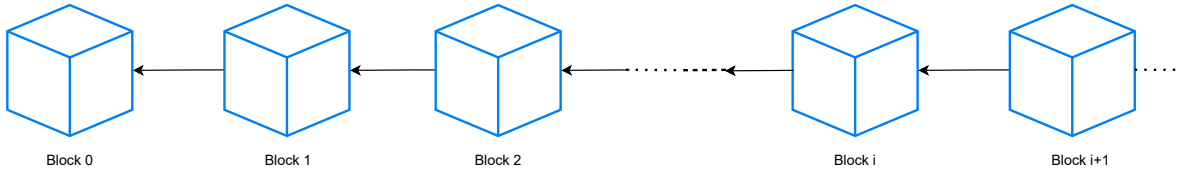

Figure 2.2: A high-level representation of the blockchain structure, where each block references the previous one, forming a continuous, immutable chain.

Block

A **block** is the fundamental unit of the Bitcoin blockchain, comprising:

- **Transaction Data**: A batch of verified transactions.
- **Block Hash**: A unique identifier obtained by applying a cryptographic hash function (SHA-256 in Bitcoin) to the block's header.
- **Previous Block Hash**: A reference that links the current block to its predecessor, ensuring continuity.
- **Nonce**: A variable that miners adjust in order to produce variant hashes of the block until one emerges that meets the Proof-of-Work (PoW) difficulty target.

Nodes and Miners

Nodes are participants in the network that verify transactions and maintain local copies of the blockchain. Among these, **miners** perform the computationally intensive task of creating new blocks. Successful miners receive compensation in the form of block rewards and transaction fees. This incentive structure aligns individual interests with the collective goal of preserving network security and reliability [26].

Proof of Work (PoW)

Proof of Work (PoW) is the consensus protocol that underpins Bitcoin's security. It requires miners to discover a hash with a specified number of leading zeros, a process governed by:

- *Network Difficulty*: Dynamically adjusted to maintain an average block creation interval of roughly ten minutes.

- *Miner Hash Rate*: The rate at which a miner can produce candidate hashes.

This computational challenge ensures that rewriting historical data is economically and computationally infeasible, thereby preserving the ledger's integrity.

Consensus

Upon the successful mining of a new block, it is broadcast to the network. Nodes validate the block by checking its cryptographic signatures, transaction authenticity, and adherence to network rules. Once accepted, the block is appended to their local copies of the blockchain. Consensus emerges naturally as honest nodes converge on the longest valid chain, referencing the latest block hash in subsequent block constructions [19].

Transactions

A **transaction** defines the transfer of bitcoin from one address to another, secured via digital signatures. After broadcast, valid transactions are relayed among nodes and eventually included in a block. Once confirmed as part of the blockchain, a transaction becomes tamper-resistant and integrated into the shared ledger's history [17].

Transaction Pool (Mempool)

The **transaction pool** (or mempool) temporarily stores valid transactions awaiting selection by miners. Miners typically prioritize transactions offering higher fees, incentivizing users to include sufficient fees to expedite confirmation.

Transaction Fees

Transaction fees compensate miners for processing and confirming transactions, ensuring that network resources are efficiently allocated. Fees are typically calculated based on transaction size (in bytes) rather than monetary value [26].

Miner Rewards

Miners are rewarded through a combination of newly minted bitcoins (the block reward) and transaction fees. This remuneration model encourages continued participation in mining activities, thereby maintaining network security and stability [18].

2.2.3 Transaction Lifecycle in the Bitcoin Network

A Bitcoin transaction undergoes the following stages, as visually represented in Figure 2.3:

1. **Creation**: A user constructs and digitally signs a transaction.
2. **Broadcast**: The transaction is disseminated across the P2P network.
3. **Validation**: Nodes verify the transaction and store it in their mempool.
4. **Mining**: Miners select transactions from the mempool, validate them, and attempt to solve the PoW puzzle to form a new block.
5. **Block Addition**: Once a miner finds a valid solution, the newly formed block (including the transaction) is appended to the blockchain.

6. **Confirmation**: Subsequent blocks building on top of this block deepen the transaction's confirmation level. After approximately six confirmations, the transaction is generally considered irreversible.

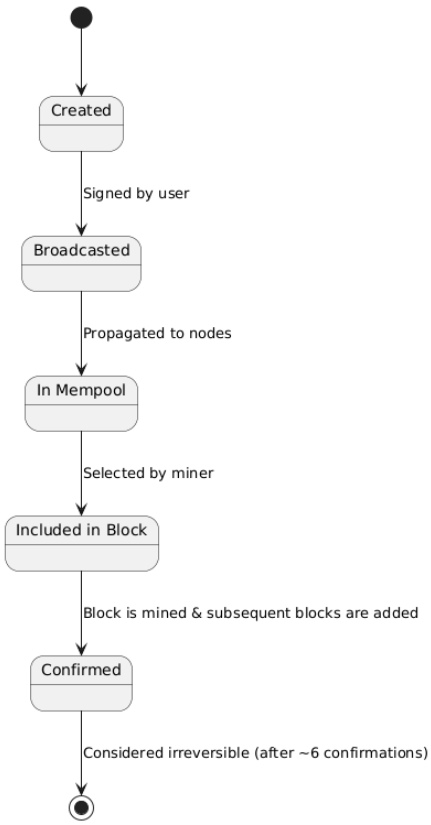

Figure 2.3: State Diagram Illustrating the Lifecycle of a Blockchain Transaction.

2.2.4 Properties of the Bitcoin Blockchain

Decentralization Bitcoin's **decentralized** architecture ensures that no single authority oversees the network. All nodes maintain local copies of the ledger and participate collectively in the consensus process, enhancing resilience and fault tolerance [18].

Security The blockchain's security derives from cryptographic hashing and the PoW mechanism. Embedding the hash of the previous block in each new block creates a verifiable chain that is computationally burdensome to modify, thereby deterring malicious actors [19].

Immutability Immutability is achieved because any attempt to alter a recorded transaction requires re-mining all subsequent blocks. The high computational costs effectively protect the ledger from retrospective modification [18].

Provenance Provenance refers to the blockchain's ability to trace all historical transactions. This transparency enables auditability and trust, as every transfer of value can be verified and followed along the chain of blocks [19].

Anonymity and Transparency All Bitcoin transactions are publicly visible, but user identities remain pseudonymous. Transactions are associated with addresses rather than personally identifiable information. Studies have shown that while this provides a degree of privacy, transaction graph analysis can reveal patterns [27]—a balance between transparency and user privacy.

2.3 Bitcoin Network Attacks

As the popularity and scale of the Bitcoin network have expanded, so too have opportunities for malicious actors to exploit potential vulnerabilities. Attackers can attempt to manipulate transactions, assert disproportionate control over mining activities, or otherwise disrupt normal network operation. In this section, we review several well-known attack vectors.

2.3.1 51% (Majority) Attack

A foundational assumption underpinning Bitcoin's security model is that no single entity can reliably control the majority of the network's computational power—also known as the hash rate. When this assumption is violated, a **51% attack** (or majority attack) becomes possible [28, 29]. In such an attack, an adversarial entity or coalition commands more than half of the network's mining resources. This dominance allows the attacker to:

- **Selectively Censor Transactions**: The attacker can refuse to include certain transactions in the blocks they mine, hindering specific users or transaction types from being confirmed.
- **Halt or Slow Down New Confirmations**: Controlling a majority of the hash power enables the attacker to repeatedly produce empty blocks or blocks containing only their chosen transactions, slowing the overall throughput of the network.
- **Reorganize the Blockchain to Enable Double-Spending**: Crucially, the attacker can secretly mine an alternative, private chain that omits previously confirmed transac-

tions. If this secret chain grows longer than the publicly accepted chain, the attacker can release it to the network, causing a reorganization (reorg) that invalidates blocks from the public chain. This capability is central to orchestrating a double-spend.

The core mechanism behind a double-spending event facilitated by a 51% attack involves constructing and deploying a competing, privately mined fork of the blockchain. By leveraging superior computational resources, the attacker can effectively "rewrite history" on the ledger. Although executing such an attack on Bitcoin is generally considered prohibitively expensive due to the immense hash power on the network, smaller and less secure blockchain implementations remain vulnerable to this form of manipulation.

Detailed Steps of Conducting a Majority-Fueled Double-Spend[1]

Identifying the Target Transaction: Suppose an attacker has already issued a transaction on the public blockchain—referred to as the *target transaction* – transferring coins to a merchant or another recipient. The attacker's goal is to erase this transaction from the blockchain and reclaim the spent funds.

Monitoring for Confirmation: Once the target transaction appears in a block on the public chain, the network begins to add subsequent blocks on top of it. Miners continue to produce blocks, each referencing the previous block's hash, thereby stacking "confirmations" on top of every included transaction. In Bitcoin, a common heuristic is to consider a transaction "safely confirmed" after six additional blocks have been mined (6 confirmations). The attacker

[1]The specific double-spending variant used in this study is detailed in the experimentation section.

waits until the target transaction is confirmed by several blocks to ensure the recipient believes the payment is final.

Privately Mining a Competing Chain: Concurrently, from the moment the attacker's target transaction first appears in a block, the attacker begins to mine an alternative, secret version of the blockchain starting from a point prior to the target transaction's inclusion. In this secret chain, the attacker either omits the target transaction altogether or includes a conflicting transaction that sends the same coins back to themselves or to another controlled address.

Because the attacker controls more than half of the hash power, they can generate new blocks at a faster rate than the honest network. This hash power advantage is critical: it ensures that the private chain, although hidden, can eventually outpace the public chain in terms of total accumulated work.

Growing the Private Chain Beyond 6 Confirmations: The attacker's aim is to surpass the length (and accumulated difficulty) of the honest chain. While the public chain reaches 6 confirmations on the target transaction, the attacker continues to extend their private fork secretly. Even though the public sees a stable blockchain with the target transaction deeply buried, the attacker's clandestine effort is on track to exceed the honest chain's length.

Releasing the Longer Private Chain: Once the attacker's secret chain becomes longer (i.e., has more accumulated proof-of-work) than the public chain, the attacker "publishes" it to the network. According to Bitcoin's consensus rules, honest nodes and miners switch to

the longest valid chain when presented with multiple competing branches. This means the previously accepted public chain—containing the target transaction—will be discarded in favor of the attacker's newly released, longer chain.

Invalidating the Target Transaction: Because the attacker's private chain does not include the target transaction, the spend that once appeared confirmed six (or more) blocks deep effectively disappears. The coins are never seen as spent in the accepted ledger's history, allowing the attacker to reuse them. For the recipient, it seems as if the funds that had been "confirmed" simply vanished, leaving them without compensation.

This process underpins the most devastating potential outcome of a 51% attack: the ability to execute double-spends with near certainty. The feasibility and cost of such an attack scale with the difficulty of outpacing honest miners, a feat generally considered prohibitive on Bitcoin itself but more attainable on smaller networks.

2.3.2 Selfish Mining Attack

Selfish mining involves withholding newly mined blocks to eventually gain a disproportionate share of the mining rewards. In this strategy, a miner or colluding group of miners [13]:

- **Private Chain Creation**: Discovers blocks and does not broadcast them, building a secret fork.
- **Strategic Release**: Releases the private chain at a time when it can invalidate honest miners' blocks, forcing them to adopt the attacker's chain.

- **Fork Resolution**: The honest network ultimately switches to the attacker's longer chain, causing honest miners to lose potential rewards and reinforcing the attacker's advantage.

This exploit erodes network fairness and may decrease trust, as it rewards strategic manipulation over honest participation.

2.3.3 Sybil Attack

A **Sybil attack** involves creating numerous fake identities (nodes) in a peer-to-peer network [30]. In Bitcoin, such attacks can:

- **Distort Network Topology**: By controlling a large number of nodes, the attacker can influence data propagation or partition honest nodes, isolating them from the network.
- **Impair Consensus and Propagation**: Flooding the network with fake nodes may slow down or interfere with the broadcast of legitimate transactions and blocks.

Bitcoin's reliance on proof-of-work and peer discovery protocols makes large-scale Sybil attacks difficult, though other smaller blockchain networks are more susceptible.

2.3.4 Network Partition Attacks

Network partition attacks exploit the underlying routing or peer-to-peer protocols to isolate a portion of the network from the rest [31, 32]. By controlling or manipulating Internet

routing or peer connections, an attacker can split the network into multiple disconnected sub-networks. Consequences include:

- **Delayed or Lost Blocks**: Nodes in a partitioned sub-network cannot receive or send new blocks to the rest of the network, causing their local chain to diverge.
- **Facilitated Double-Spend**: Attackers can exploit this partition to confirm transactions in an isolated segment of the network, then release a longer chain to the main network that omits those transactions.
- **Reduced Hash Power for Honest Miners**: If significant mining power is stuck in the smaller partition, it becomes easier for attackers in the main partition to execute chain reorganizations.

Such attacks demonstrate that controlling the flow of information in a decentralized network can be as powerful as controlling a majority of the hashing power. Although Bitcoin implements mechanisms to mitigate these attacks (e.g., multiple peer connections and random node discovery), partial partitions or Eclipse attacks remain a real threat, especially for less robust networks.

In summary, Bitcoin's design provides robust security under honest conditions, but these attacks underscore the nuanced vulnerabilities that surface when adversaries gain control, manipulate network resources, or undermine consensus assumptions. It is important to note that all attacks against Bitcoin are attacks against transaction *finality*, i.e., either delay or completely deny the finality of honest transactions at an individual or massive scale,

or, conversely, promote the finality of dishonest transactions. Through careful study and simulation, researchers and developers can better understand these threats and implement improvements to bolster the resilience of Bitcoin and other blockchain networks.

2.4 Blockchain Simulators

With blockchain applications now extending beyond cryptocurrencies into sectors such as supply chain management, healthcare, and the Internet of Things (IoT), the need for robust tools to evaluate and simulate blockchain performance has become increasingly critical [15, 16, 29]. One of them is *blockchain simulators*, which reproduce the operation of blockchain protocols in controlled environments and conditions, allowing the generation of data useful for assessing the performance of the corresponding protocol designs.

2.4.1 Importance of Blockchain Simulation

Simulating blockchain networks allows researchers and developers to test, validate, and improve their systems in a controlled environment, without the high costs and risks associated with real-world deployments. Simulators provide valuable insights into the behavior of blockchain networks, including key performance metrics such as:

- **Transaction Throughput**: The rate at which transactions are processed and confirmed in the network.
- **Latency**: The time taken for a transaction to be confirmed and added to the blockchain.

- **Consensus Mechanisms**: The efficiency and robustness of different consensus protocols, such as Proof of Work (PoW) or Proof of Stake (PoS).

- **Network Security**: The network's resistance to attacks such as double-spending, 51% attacks, or selfish mining [13, 33].

Blockchain simulators also allow developers to experiment with network topologies, node behaviors, and new consensus mechanisms under different conditions, making them invaluable tools for performance evaluation, scalability testing, and security analysis [34, 35, 36].

2.4.2 Key Blockchain Simulators and Their Approaches

A variety of blockchain simulators exist to help researchers and practitioners study consensus protocols, network behavior, and performance metrics under controlled, reproducible conditions. Broadly, these simulators can be grouped into two main categories: *abstract/model-based simulators* and *native/implementation-based simulators*. Each category balances simulation fidelity, scalability, and complexity differently, making the choice of simulator highly dependent on the specific experimental goals.

Abstract/Model-Based Simulators. Model-based simulators re-implement blockchain logic in simplified or controlled environments. Instead of running a full node software stack, they typically replicate essential components such as networking, consensus, and block/transaction processing in a more abstract form. These abstractions allow for:

- **Easy Modifiability and Extensibility:** The consensus logic, mining strategies, or

network topology can be readily swapped or modified, enabling rapid prototyping and comparative studies of protocol variants.

- **Reduced Resource Requirements:** Since they emulate only the relevant mechanisms (e.g., PoW, PoS) with minimal overhead, they can scale to large numbers of simulated nodes without demanding extensive compute resources.

- **Focused Analysis:** By stripping away non-essential details of real blockchain clients, model-based simulators allow researchers to isolate specific metrics (e.g., throughput, propagation delay, or fork rates) and systematically evaluate their sensitivity to varying parameters.

However, these advantages come at the cost of lower fidelity. Model-based simulators may neglect subtle interactions or edge cases present in actual blockchain clients (e.g., real networking stacks, protocol versions, or peer discovery mechanics). Consequently, model-based results often need to be cross-validated in more realistic environments before production-grade conclusions can be drawn.

Native/Implementation-Based Simulators. On the other end of the spectrum, native simulators run unmodified or minimally modified blockchain clients (e.g., Bitcoin Core, Geth) in containerized or virtualized setups. This approach offers:

- **High Fidelity:** By executing real node software, native simulators capture all protocol intricacies (e.g., P2P layer, mempool dynamics, wallet/UTXO management) that might be overlooked in model-based approaches.

- **Realistic Network Behavior:** Because networking components are not re-implemented, peer discovery, block gossip, and synchronization proceed exactly as in a real-world deployment.

- **Identification of Implementation Bugs:** Subtle client-side bugs or version-specific inconsistencies are more likely to surface under realistic conditions.

Nonetheless, native simulations are resource-intensive, potentially involving hundreds or thousands of containers/VMs to replicate large-scale networks. This overhead can limit experimentation to fewer protocol variants or smaller time windows, and often requires specialized infrastructure or cloud-based orchestration.

Selected Simulators. The following simulators illustrate the diversity of approaches, ranging from purely abstract to fully native:

- **BlockSim (Model-Based)**: An extensible Python simulator for Bitcoin and Ethereum [37]. Its modular design supports detailed analyses of throughput, propagation delay, and mining strategies. BlockSim's simplicity and flexibility make it suitable for researchers interested in experimenting with different consensus adjustments or network assumptions.

- **SIMBA (Model-Based)**: Specializes in Bitcoin's PoW consensus, emphasizing security metrics like double-spending probabilities under various network topologies [38]. It is particularly useful for examining how network delays or changes in hashrate distribution affect the likelihood of successful attacks.

- **BlockEval (Model-Based)**: Integrates deep learning techniques for private (enterprise) blockchain performance assessment [39]. By incorporating machine learning models, it can predict performance bottlenecks and optimize resource allocation strategies for permissioned or private chain deployments.

- **SimBlock (Model-Based)**: Focuses on node behaviors and geographic network latency, providing insights into block propagation efficiency [40]. Researchers can adjust node distributions or communication parameters to analyze how real-world latency factors might impact fork rates or block propagation times.

- **VIBES & eVIBES (Hybrid Visualization)**: Provide real-time visualization and large-scale simulation capabilities for Bitcoin and Ethereum, respectively [41, 42]. These frameworks are particularly valuable for demonstrating how blocks and transactions flow across the network, serving both educational and diagnostic purposes.

- **JABS (Model-Based)**: A high-performance simulator supporting various consensus algorithms (PoW, PoS, DPoS) [43], making it suitable for side-by-side comparisons of performance, security, and resource usage in both permissioned and permissionless settings.

- **BlockCompass (Native/Implementation-Based)**: A benchmarking tool designed to evaluate the performance of various blockchain platforms, including Hyperledger Fabric, Hyperledger Sawtooth, and Ethereum [44]. It measures multiple performance indicators such as resource consumption (CPU, memory, network I/O), latency, trans-

mission rate, throughput, and error rate. BlockCompass provides real-time charts and the ability to export detailed reports, offering high fidelity insights into how a blockchain platform performs in real-world scenarios.

- **Native Testnets / Cloud-Based Emulators (Implementation-Based)**: Run multiple real clients (e.g., Bitcoin Core, Geth) in containerized environments to capture protocol behavior exactly. While highly resource-intensive, this approach offers unmatched fidelity for studying client-specific issues, testing new forks or upgrades, and observing emergent behaviors that may not be visible in abstract simulations.

In summary, the choice between a model-based and an implementation-based simulator depends on the research question, available infrastructure, and desired fidelity. Model-based simulators enable quick experimentation and large-scale scenario testing at relatively low cost, but may leave out critical real-world complexities. Implementation-based simulators, while resource-heavy, can uncover subtle bugs and dynamics inherent to actual clients—making them indispensable for final-stage validation or production-focused investigations.

2.4.3 Simulators vs. Simulation Frameworks

Blockchain simulators are invaluable tools for performance evaluation, security analysis, and consensus mechanism testing. They provide a safe, cost-effective environment for exploring the behavior of blockchain networks under a wide range of conditions. Simulators like BlockSim, SIMBA, SimBlock, and others enable researchers and developers to model complex blockchain systems and study their resilience to attacks, helping to advance the field of blockchain

technology.

However, in all the simulators we reviewed, there is little guidance on how they should be used. Current work on blockchain simulation focuses largely on qualities like efficiency, extensibility, modularity, or architectural transparency. There is little or no direction as to which specific measurements each simulator provides that would allow protocol researchers to perform systematic evaluations, conduct comparative analyses between protocols, or even evaluate multiple variations of the same protocol under different conditions [29, 16, 45, 46, 47].

For example, BlockSim [37] provides a stochastic model for analyzing transaction confirmation times and orphan rates but lacks a comprehensive framework for evaluating other aspects like network security or consensus efficiency. SIMBA [38] focuses on network layer simulation and allows for modeling different network topologies, but does not offer specific metrics or methodologies for comparing different consensus algorithms. SimBlock [40] offers a modular architecture for simulating various consensus mechanisms and network attacks, but leaves the definition and interpretation of evaluation metrics largely to the user. Similarly, BlockEval [39] focuses on energy consumption of different consensus protocols, providing specific metrics for this aspect, but not a broader evaluation framework. VIBES and eVIBES [41, 42] offer a flexible foundation for simulating blockchain systems with customizable components, yet they do not prescribe systematic metrics or analysis techniques. In demonstrative experiments of existing simulators, ad-hoc constructs are introduced such as the fork-rate, i.e., the proportion of blocks that do not make it to the main chain, and hence contain non-final transactions.

As much such constructs are useful for studying Bitcoin and linear blockchain-based Proof-of-Work, they are not easily applicable to competing consensus systems that, e.g., do not allow for branching in the first place, or that do not employ linear blockchain structures but e.g., Directed Acyclic Graphs (DAGs) [5, 48].

In contrast, a few efforts toward standardized evaluation have emerged in the literature. Frameworks like BlockBench [45] and Hyperledger Caliper [46] attempt to provide benchmarking methodologies and metrics for assessing performance aspects such as throughput, latency, and scalability, albeit mainly for private and enterprise-focused blockchain platforms. Systematization-of-Knowledge (SoK) studies [29, 34] and surveys [15, 16] emphasize the need for commonly accepted metrics, guidelines, and benchmarks to facilitate meaningful comparisons. Efforts by standardization bodies, such as ISO/TC 307 and NIST [47], also highlight the importance of common terminology and benchmarking methods. However, these initiatives remain limited, and there is no widely adopted, comprehensive framework that simultaneously addresses performance, security, scalability, and energy efficiency in a standardized manner.

Hence, building a well-performing simulator is only part of a simulation-based analysis. The remaining and more challenging part involves determining how to use simulations to generate data whose analysis can yield insights about protocol quality and allow for systematic comparisons. Recognizing these gaps, the next chapter introduces CNSim, a simulator and evaluation framework specifically designed to fill this void, offering a structured approach to protocol assessment and comparison. By providing both simulation capabilities and a

methodology for measuring key metrics, CNSim aims to advance the state-of-the-art in blockchain protocol evaluation.

Chapter 3

CNSim: Finality-Based Blockchain Protocol Evaluation

3.1 Finality-based Blockchain Network Evaluation

As we saw in the previous chapter, several proposals for simulating blockchain protocols have been proposed in the literature. The focus of most of these proposals is advancing the simulation process itself, including its scalability (ability to simulate large networks), efficiency (quick simulation of many events), granularity and/or abstraction (simulating protocol details vs. abstracting details for efficiency,) and modularity and extensibility. However, relatively little effort is dedicated towards understanding how a simulator can be systematically used to study modeled protocols and behaviors. In particular, there is a need for a theoretical framework that translates the potentially vast data that a blockchain simulator produces into metrics that allow not only assessment of a single protocol and behavior, but, importantly, comparison among protocols that are often vastly different in design and philosophy. This thesis adopts a *finality-based* approach to blockchain performance evaluation, which we

describe in this section.

3.1.1 Overview

As we saw, blockchain networks, such as Bitcoin, rely on a decentralized set of nodes to verify transactions through a *consensus protocol* [1]. Consensus protocols are the foundation of these networks, ensuring that all participants agree on the validity and ordering of transactions. They differ in terms of the techniques used to validate transactions and secure the network—from computationally intensive Proof of Work (PoW) [19, 20] to resource-staking Proof of Stake (PoS) [21, 22, 23, 24] or even more specialized protocols—but their primary goal remains the same: to ensure that a transaction is accepted as valid and, ideally, becomes irreversible (or highly unlikely to be reversed) within a reasonable time frame. This property is commonly referred to as *transaction finality*.

Transaction finality is crucial for blockchain networks because it determines the point at which a transaction can be trusted to be permanent and immutable. The ability to quickly and confidently declare a transaction final has significant implications for both the performance and the security of the system. From a user perspective, a transaction that achieves strong finality can be treated like an irreversible payment, reducing the risk associated with delayed settlement or potential rollbacks.

3.1.2 Consensus Protocols and Finality

Different consensus protocols handle finality in varying ways. In Bitcoin's PoW, finality is *probabilistic*: as more blocks are appended to the chain, the probability that any particular transaction will be reversed diminishes – but never becomes zero. After a certain number of confirmations – often cited as six blocks in Bitcoin – the odds of an attacker successfully reorganizing the chain and removing a transaction become negligible [1], effectively making the transaction final for most practical purposes [19, 20].

On the other hand, *forced finality* is offered by certain consensus mechanisms, typically found in permissioned blockchains or specific PoS protocols [21, 22, 23, 24]. Here, a transaction can be considered final as soon as it appears in a block that has been irrevocably committed by a quorum of nodes. This removes the need for multiple confirmations, potentially enabling near-instant finalization. Such protocols often rely on Byzantine Fault Tolerant (BFT) consensus algorithms [49], which provide mathematical guarantees of finality once a sufficiently large fraction of nodes have agreed. Nevertheless, none of these properties protects the protocol from malicious participants, who can intrude and disrupt any open-access ("public") blockchain, independent on consensus protocol followed.

3.1.3 Formalizing Finality: Degrees of Belief and Consensus Thresholds

From informal intuition to temporal-logic clauses. Let us now attempt a formal definition of finality, which will later allow us to derive appropriate measures of it based

on simulation data. The definition is based on interpreting finality as an *from-there-and-on* property [50]. We can formally express the property in real-time Metric Temporal Logic (MTL). Specifically, let f be a proposition about a fact in the world, including that a transaction has taken place. With A also denote the set of witnessing agents (network nodes) that are considered to be actively participating in the witnessing proceedings. As such, the membership of A changes in time as existing agents depart and new agents join. Using the standard "eventually" ($\Diamond$) and "always" ($\Box$) modalities, and using $\Diamond_{<t}$ to denote "sometime within the next t time-units", finality is defined as:

$$\text{final}(A, f, t, d) \equiv \Diamond_{<t}\, \Box\big(\text{degBelief}(A, f) \geq d\big), \tag{3.1}$$

where $\text{degBelief}(A, f) \in [0, 1]$ is the community's collective belief in f (defined below) and $d \in (0, 1]$ is an application-chosen consensus threshold. Equation (3.1) reads: *"proposition f is final iff the community's belief in f exceeds d within time t and never again sinks below d thereafter"*

Quantifying collective belief. Let us next focus on how we quantify a community's collective belief. We first assume that each node $n \in A$ maintains a local conviction $\text{believes}(n, f, t) \in [0, 1]$ that f is valid at time t. For classical chain-based protocols, believes $\in \{0, 1\}$ – a block either sits on the node's canonical chain or it does not. DAG or hybrid ledgers (e.g., IOTA's Tangle) yield fractional convictions derived from random-walk tip-selection probabilities.

The *degree of belief* of the whole community at time t is then a simple average:

$$\text{degBelief}(A, f, t) = \frac{1}{|A_t|} \sum_{n \in A_t} \text{believes}(n, f, t), \tag{3.2}$$

with $A_t = \{\, n \in A \mid n \textit{ active at time } t \,\}$. Thus, degBelief is precisely "the probability that a randomly chosen active witness currently accepts f".

From Boolean property to stochastic metric. Given a history and whether / the degree to which proposition f is believed to be true by each of the active nodes, the predicate $\text{final}(A, f, t, d)$ is deterministically either true or false. However, we are interested in assessing whether f will be final at a time where we do not have its history yet. Hence it is more useful to talk about the probability of f becoming final:

$$\text{finality}(A, f, t, d) \equiv P\big(\text{final}(A, f, t, d)\big), \tag{3.3}$$

called the *finality metric*. It answers the practical question: *"How likely is it that our transaction will reach irrevocable consensus by community A within t seconds subject to belief threshold d ?"*.

Empirical estimation of finality. Let us now see how we can estimate the above probability via simulation. Consider time t_0 in the operation of a consensus network (simulated or otherwise) when a proposition f arrives at the network. Let us obtain a snapshot of the network at that time t_0 and create a set K of copies thereof. Using the snapshot as the

initial condition, resume the proceedings of the network in each copy in K independently and up to a common time horizon t_h. In other words, we acquire K independent continuations s of the network state taken at the arrival time t_0 of a reference transaction f. Let also $\chi : \{true, false\} \mapsto \{0, 1\}$, i.e., $\chi(p) = 1$ if p is $true$ and $\chi(p) = 0$ otherwise. It follows then from the definition of final that:

$$\text{final}_{t_h}^{(s)}(A, f, d) = \chi\Big(\exists\, t_1 \leq t_h \,\text{s.t. degBelief}^{(s)}(A, f, t_1) \geq d \wedge \\ \text{degBelief}^{(s)}(A, f, \tau) \geq d,\ \forall\, \tau \in [t_1, t_h] \Big),$$

where t_h is a time-horizon of interest, $\text{degBelief}^{(s)}(A, f, t)$ is the degree of belief of community A in proposition f at time t in history s. Then, $\text{final}_{t_h}^{(s)}(A, f, d)$ is 1 iff f is final subject to A, d and h, and 0 otherwise. Then, the unbiased maximum-likelihood estimator is the proportion of times among the K histories that f turned out to be final:

$$\widehat{\text{finality}}_{K,t_h}(A, f, d) = \frac{1}{K} \sum_{s=1}^{K} \text{final}_{t_h}^{(s)}(A, f, d)$$

converges to the true probability (3.3) for large K.

The above offers us a concrete metric for measuring finality via simulation. First, we sample one transaction from an incoming workload of such. We simulate the network until the time the sample arrives at the system. This is t_0 and the consensus network is in a particular state. Starting always from that state, we simulate the network K independent times up to

time t_h. At the end of each such simulated episode we check to see if final(A, f, t, d).

Interpreting the parameters d and t. Neither d nor t is fixed by a protocol; both are *service-level objectives* set by the application domain. For example:

- **Retail PoS.** A shopkeeper may insist on $d = 0.999$ ("one in a thousand chance of reversal") but can tolerate $t \approx 5\,\text{s}$ for a smooth checkout.
- **High-frequency micro-transaction trading.** A trading desk might require $t < 500\,\text{ms}$ yet accept $d = 0.95$, relying on post-trade netting for residual risk.

Parameters will be, thus, fixed to the requirements of the analysis at hand.

3.1.4 Worked Miniature Example

To illustrate this framework, we adapt the demonstrative example from our technical report [50]. Consider a five-node network where a transaction of interest, f, is introduced at time t_0. We want to assess its finality with a service-level objective of a consensus threshold $d = 0.9$ over a defined time horizon t_h.

To estimate the finality probability, three independent simulations ($K = 3$) are run from an identical network snapshot taken at t_0. The schematic in Figure 3.1 shows the outcome for each node's belief state over time in the three runs.

Finality Calculation The outcome for each simulation run is assessed against the finality criteria:

- **Simulation 1 (s_1):** The group belief, `degBelief`, surpasses the 0.9 threshold at time t_4 and remains at or above it until the time horizon t_h. Therefore, the transaction achieves finality in this run.

- **Simulation 2 (s_2):** The `degBelief` surpasses the 0.9 threshold at t_3 but then drops below it at t_4. This pattern can emerge in an eventual-consensus network if a fork occurs and the chain containing the transaction is orphaned. Since the belief was not perpetually maintained, the transaction does *not* achieve finality.

- **Simulation 3 (s_3):** The `degBelief` never reaches the 0.9 threshold within the time horizon t_h. The transaction, therefore, does not achieve finality in this run.

Out of the three simulations, only one was successful. The estimated finality is therefore calculated as:

$$\widehat{\text{Finality}}_{3,t_h}(f, 0.9) = \frac{1}{3} \approx 0.33$$

This result provides a concrete, quantitative measure of the network's performance for this specific scenario and set of service-level objectives.

Confidence Plots The data from these simulations can be visualized using a **belief evolution graph**, as shown in Figure 3.2. This visualization provides not just a single finality number but also qualitative insights into how consensus forms—or fails to form—over time in the network. By grounding the evaluation in this formal, simulation-driven framework, we can move beyond ad-hoc rules of thumb like "6 confirmations" and towards a more rigorous, comparative science of blockchain networks.

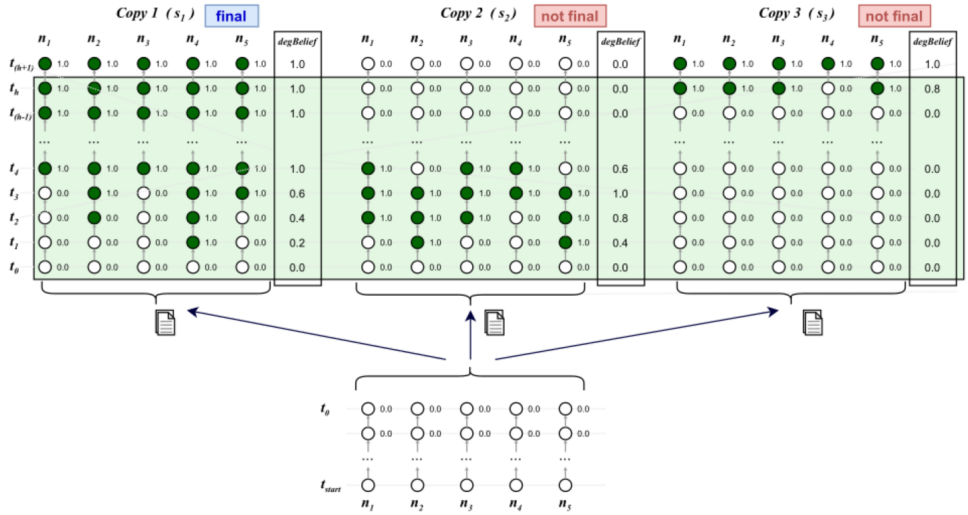

Figure 3.1: A schematic showing three independent simulation copies (s_1, s_2, s_3) evolving from a common state at t_0. Each node's binary belief (0 or 1) is shown, along with the calculated `degBelief` at each time step.

Having established the theoretical framework for measuring protocol performance via the finality metric, the remainder of this chapter details CNSim, the simulation platform designed to implement this methodology. We will now explore the core architecture of CNSim and its specific implementation of the Bitcoin protocol, which provides the foundation for the attack modeling discussed later.

3.1.5 CNSim: A Comparative Simulation Platform

In this research, CNSim is employed as the simulation platform. CNSim is aimed at being highly configurable and able to simulate a wide range of consensus protocols and network conditions. It supports the injection of adversarial behaviors and can track the evolution of degBelief(f, t) over time, facilitating in-depth finality analyses.

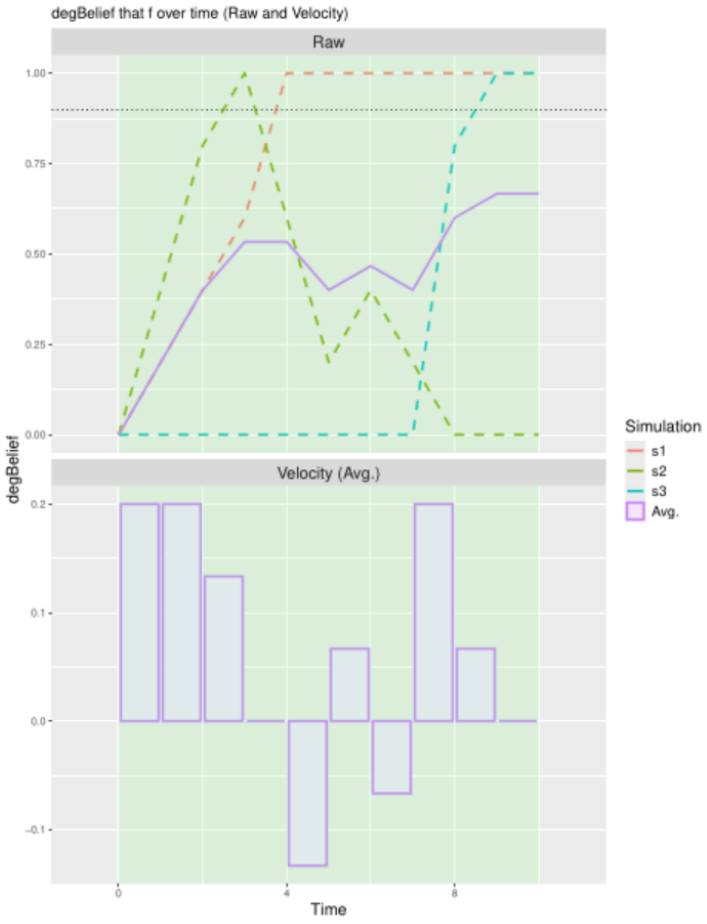

Figure 3.2: The corresponding belief evolution graph visualizes the `degBelief` over time for each simulation (dashed lines) and their average (solid line). The lower plot shows the velocity (rate of change) of the average belief, where negative values can indicate a loss of consensus, such as a fork.

CNSim enables fine-grained experimentation with variables such as:

- **Node Distribution and Topology:** Different levels of decentralization, such as varying node counts or heterogeneous network graphs.
- **Latency and Reliability:** Configurations to explore how communication delays or unreliable links affect finality.

- **Attack Scenarios:** Systematic testing of consensus protocols against adversarial power ratios, double-spending attempts, eclipse attacks, or other strategic threats.

3.2 The CNSim Simulation Architecture

CNSim is a flexible and extensible simulation framework designed to model the dynamics of Bitcoin and related blockchain networks. At its core, CNSim leverages an event-driven simulation engine to capture the asynchronous and stochastic behavior inherent in real-world networks. Its architecture is organized around several interrelated components, each addressing distinct aspects of the simulation process. We will review each component in more detail in the next sections.

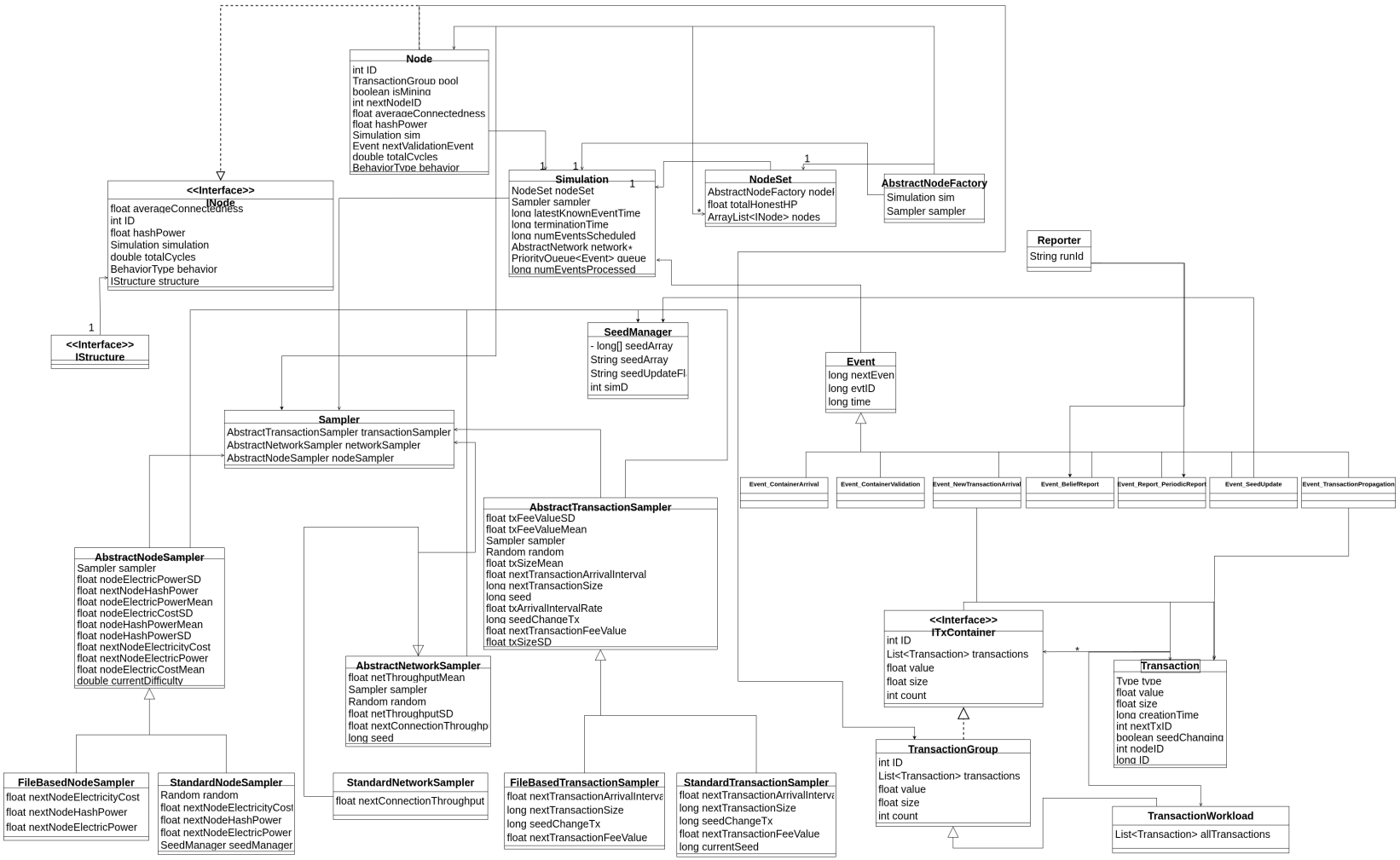

Figure 3.3: Diagram illustrating the overview architecture of the engine.

3.2.1 Simulation Queue and Event Processing

The simulation adopts an **event-driven execution model**, where time advances only when relevant system changes occur. Central to this model is the **Simulation Queue**—a priority queue that orders events by their timestamps. By processing events chronologically, the simulation preserves logical causality, respects network delays, and avoids unnecessary computations during idle periods.

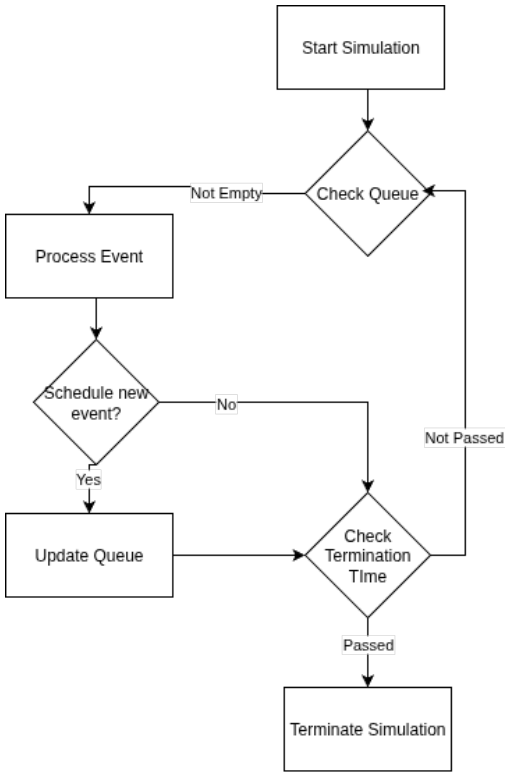

Figure 3.4: Diagram illustrating the simulation queue and event processing mechanism.

Simulation Queue as a Time-Ordered Scheduler At each simulation step, the earliest event is dequeued and executed, potentially generating additional events that are reinserted with updated timestamps. This mechanism ensures a deterministic sequence of actions,

allowing the simulation clock to move forward only when meaningful activity takes place.

Temporal Progression and Event Handling Because events can trigger subsequent state changes, the priority queue dynamically reorders itself as new events arise. Detailed information about the types and structure of these events is provided in the following subsection.

3.2.2 Event Hierarchy

The **event-driven engine** is the core mechanism that propels the simulation forward. Each notable state change—such as a transaction's dissemination, the mining of a new block, or the validation of an existing chain—is encapsulated as an event. A generic `Event` class underpins this system, serving as the abstract parent from which specialized event types inherit. These events are prioritized by their scheduled execution times and stored in a priority queue, ensuring that simulated time advances only when a consequential state change occurs.

At the foundation of this engine is the abstract `Event` class, which defines common attributes and behaviors for all event types. Subclasses extend this base class to capture specific occurrences, creating a clear and modular design. Broadly, the event hierarchy can be grouped into three categories:

1. **Node-Related Events:**

   These events orchestrate interactions among nodes in the network, capturing the distributed nature of blockchain systems.

- `Event_NewTransactionArrival`: Denotes the first arrival of a new transaction at a node. *(Parameters: transaction object, node object, timestamp).*
- `Event_TransactionPropagation`: Denotes the arrival of a transaction to a node as a result of propagation from another node. *(Parameters: transaction object, receiving node object, timestamp).*
- `Event_ContainerArrival`: Represents the arrival of a container (e.g., a block) at a node. *(Parameters: container object, receiving node object, timestamp).*
- `Event_ContainerValidation`: Marks the time when the validation of a node, which, e.g., in PoW networks, may be a lengthy mining run, is complete. *(Parameters: container object, validator node object, timestamp).*

2. **Reporting Events:**

   Periodic data gathering and system monitoring are facilitated by these events, which record snapshots of network state at specified intervals.

   - `Event_Report_BeliefReport`: Requests the recording of each node's view (or "belief state") regarding the validity of a specific sample of transactions. *(Parameters: timestamp).*
   - `Event_Report_NodeStatusReport`: to log their status, including connectivity metrics and performance indicators. *(Parameters: timestamp).*
   - `Event_Report_PeriodicReport`: Requests the nodes to log regular snapshots of global system conditions. *(Parameters: timestamp).*

- `Event_Report_StructureReport`: Requests a node to save the status of their local structure (e.g., the blockchain). *(Parameters: timestamp).*

3. **Auxiliary Events:**

   These events offer additional support functions that enhance the simulation's accuracy and adaptability.

   - `Event_SeedUpdate`: Refreshes the simulation's random seed values at specified intervals, ensuring consistent stochastic behavior. *(Parameters: sampler object, timestamp).*

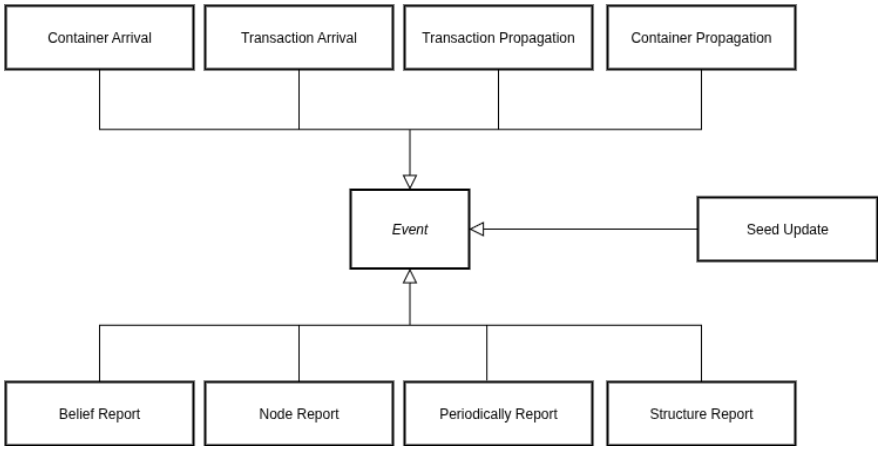

Figure 3.5: Diagram illustrating the simulation queue and event processing mechanism.

All events are ordered in a priority queue by means of an `EventTimeComparator`, which guarantees that earlier events are processed first. This approach ensures deterministic progression, with the simulation clock moving forward only when significant actions occur. The modular structure of this event system also supports straightforward extension, allowing for new event types to be integrated seamlessly.

3.2.3 Node Architecture Components

The node architecture consists of four primary components:

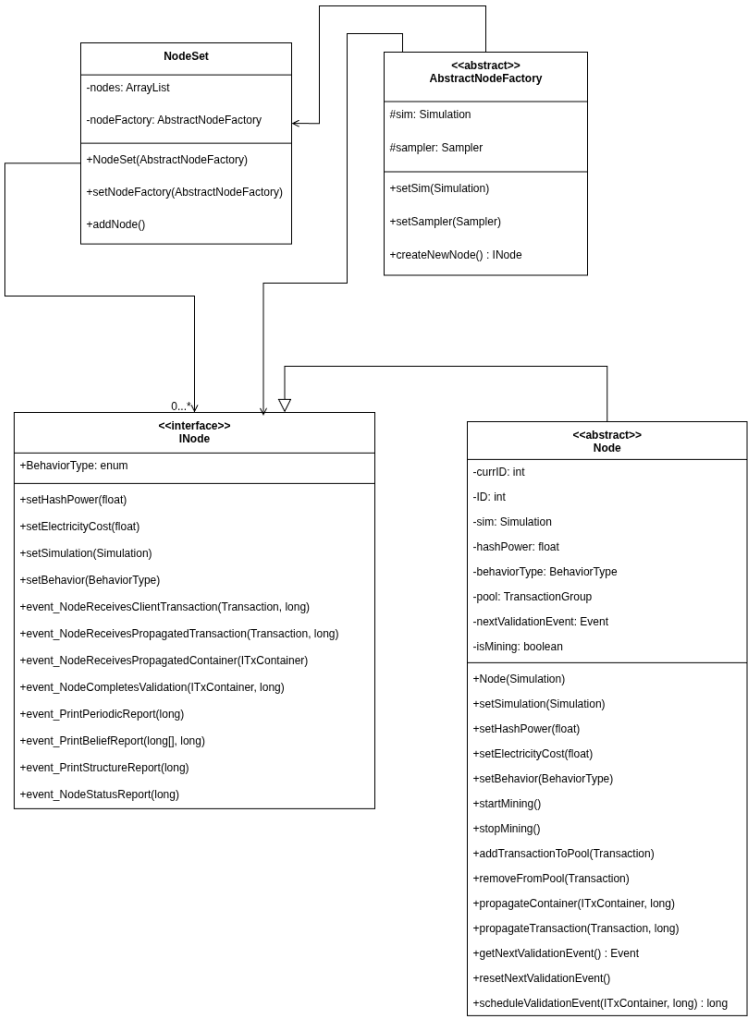

Figure 3.6: Diagram illustrating the node architecture in engine.

- **Node Interface (`INode`)**

  The `INode` interface serves as the foundational contract for all node implementations in the simulation. This interface defines the essential behaviors and capabilities that any node in the network must possess, including event handling methods that implement

the core protocol logic. Through this interface, nodes can respond to various network events such as transaction propagation, container validation, and structure updates, effectively encoding the blockchain protocol's rules and behaviors.

- **Node Implementation**

  The concrete `Node` class implements the `INode` interface, providing the default behavior for network nodes. This implementation includes:

  - Event-driven state management through four primary transaction and container events:
    - `Event_TransactionArrival`: Handles incoming transactions from the network
    - `Event_TransactionPropagation & Event_ContainerPropagation`: Manages the propagation of transactions or containers to other nodes
    - `Event_ContainerArrival`: Processes incoming block containers from the network
    - `Event_ContainerValidation`: Handles the validation of block containers
  - Local blockchain structure maintenance and updates through event handlers that modify the node's local state
  - Transaction pool management and validation logic, including:
    - Transaction addition and removal from the pool
    - Validation of transaction integrity

    - Management of transaction dependencies
  - Mining and consensus mechanism implementation through:
    - Block creation and validation
    - Hash power management
    - Consensus rule enforcement
  - Protocol-specific behavior through event handlers that implement:
    - Network message processing
    - State synchronization
    - Conflict resolution

- **Node Factory**

  The `AbstractNodeFactory` implements the Factory pattern, providing a standardized mechanism for node creation. This abstraction layer enables:

  - Consistent node instantiation with protocol-specific configurations
  - Runtime configuration of node properties and behavior types
  - Easy extension for different protocol implementations
  - Customization of event handling and structure update mechanisms

- **Node Collection Management**

  The `NodeSet` class manages collections of nodes, handling:

  - Group operations on nodes and their local structures

- Network-wide event propagation and synchronization
- Efficient batch processing of protocol events
- Network topology maintenance and node relationship management

The entire logic of the blockchain protocol is implemented through the node's event handling system. Each node maintains its own local copy of the blockchain structure and responds to various events such as transaction arrivals, block validations, and network messages. This event-driven architecture allows for flexible protocol implementation while maintaining consistency across the network. The protocol's rules and behaviors are encoded in how nodes react to these events and update their local structures, making the system both modular and extensible.

3.2.4 Sampler Design

The Sampler in CNSim is designed to systematically manage and generate values essential for simulating transaction behaviors, node characteristics, and network conditions. Located within the `ca.yorku.cmg.cnsim.engine` package, it follows a modular architecture by integrating three specialized abstract samplers:

- **AbstractTransactionSampler**: Manages the sampling of transaction-specific parameters, including arrival rates, transaction sizes, and values.
- **AbstractNodeSampler**: Responsible for generating node-related attributes such as computational hash power, energy consumption, and operational costs, ensuring

realistic node behaviors.

- **AbstractNetworkSampler**: Handles network-related sampling tasks, particularly network throughput and connectivity parameters.

Each abstract sampler has two primary concrete implementations to cater to diverse simulation needs:

- **Standard Samplers**: Utilize statistical distributions (e.g., Gaussian, Poisson) to generate realistic and probabilistic simulation values.
- **FileBased Samplers**: Employ predefined datasets or configuration files, enabling deterministic and reproducible simulation scenarios.

The Sampler class offers flexibility and encapsulation through getter and setter methods, allowing dynamic assignment of sampler implementations according to simulation requirements. Additionally, it provides general methods for common sampling tasks, such as generating Poisson intervals, Gaussian distributions, and random integer ranges.

This design emphasizes extensibility, enabling users to develop specialized sampling strategies tailored to specific simulation scenarios effectively.

3.2.5 IStructure Interface

The `IStructure` interface provides a generic model for representing and managing transaction finality within different types of network structures in the simulation. Its primary role is to define the fundamental operations required for determining if a transaction is part of a

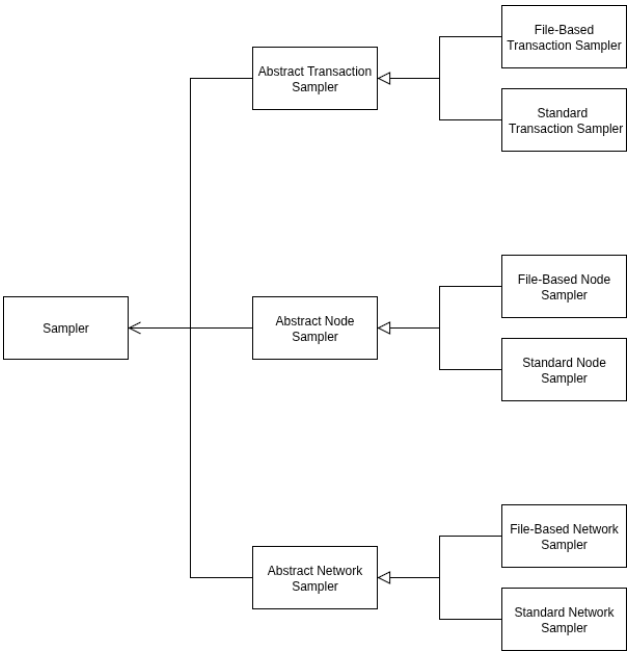

Figure 3.7: Hierarchical and modular design of the Sampler class and its interactions with associated abstract and concrete samplers.

given structure (e.g., a blockchain or a Directed Acyclic Graph), ensuring consistency and integrity across various implementations. Specifically, `IStructure` declares methods such as `transactionInStructure(long txID)` to verify the inclusion of a transaction, enabling uniform checks regardless of the underlying data model. This design promotes flexibility by allowing the same interface to be implemented by multiple structures (e.g., a linear chain, a DAG) while preserving a consistent interaction pattern. As a result, the simulation framework can seamlessly accommodate diverse consensus approaches and finality rules by integrating new classes that conform to the `IStructure` contract.

3.2.6 Reporter Module

The **Reporter** module in the blockchain simulator project is responsible for systematically logging simulation activities, events, and relevant data to facilitate comprehensive analysis and reporting. It operates through static methods, ensuring centralized and consistent management of log entries throughout the simulation lifecycle. Specifically, the Reporter performs the following tasks:

- **Event Logging:** Captures detailed records each time an event occurs during the simulation, noting event type, simulation and system timestamps, involved nodes, and related objects.
- **Transaction Logging:** Tracks incoming transactions, logging attributes such as transaction size, value, and arrival time, allowing for transaction flow analysis.
- **Node Reporting:** At the conclusion of the simulation, it records essential node metrics including computational power, electrical consumption, and economic factors, enabling evaluation of node performance and efficiency.
- **Network Events and Beliefs:** Records network configuration changes and nodes' beliefs about transaction validity, providing insights into network dynamics and consensus behaviors.
- **Error Handling:** Maintains an error log to document unexpected behaviors or issues encountered during simulation runs.

Each log is stored in a clearly formatted CSV file, tagged with simulation identifiers and timestamps, enabling straightforward post-simulation analysis and troubleshooting.

3.2.7 Configuration Modules (Config and ConfigInitializer)

The **Config** and **ConfigInitializer** modules manage the initialization, loading, and validation of configuration settings for the simulator. Specifically, their roles include:

- **Configuration Loading:** `ConfigInitializer` combines command-line arguments with configuration file properties, prioritizing command-line inputs to allow flexible adjustments.
- **Property Management:** The `Config` class provides centralized access methods to retrieve configuration properties in various data types (string, integer, long, float, double, boolean), ensuring consistent and type-safe configuration handling throughout the simulator.
- **Configuration Validation:** Both modules include robust validation mechanisms to verify the existence and correctness of configuration settings, essential files, and directories, helping prevent runtime errors due to misconfigurations.
- **Utility Parsing Methods:** They include specialized methods for parsing complex configuration strings into usable arrays (e.g., lists of IDs or booleans), enhancing configurability and simplifying simulation setup.

Overall, these modules ensure reliable, flexible, and error-resistant configuration management, critical for conducting accurate and repeatable simulations.

3.3 Bitcoin Network Implementation in CNSim

CNSim's Bitcoin network simulation module is a specialized extension of the core simulation engine that focuses on modeling Bitcoin's protocol dynamics, mining operations, and blockchain consensus mechanisms. This section describes how Bitcoin-specific functionalities are integrated into the framework. Figure 3.8 illustrates the high-level architecture and component interactions unique to the Bitcoin simulation module.

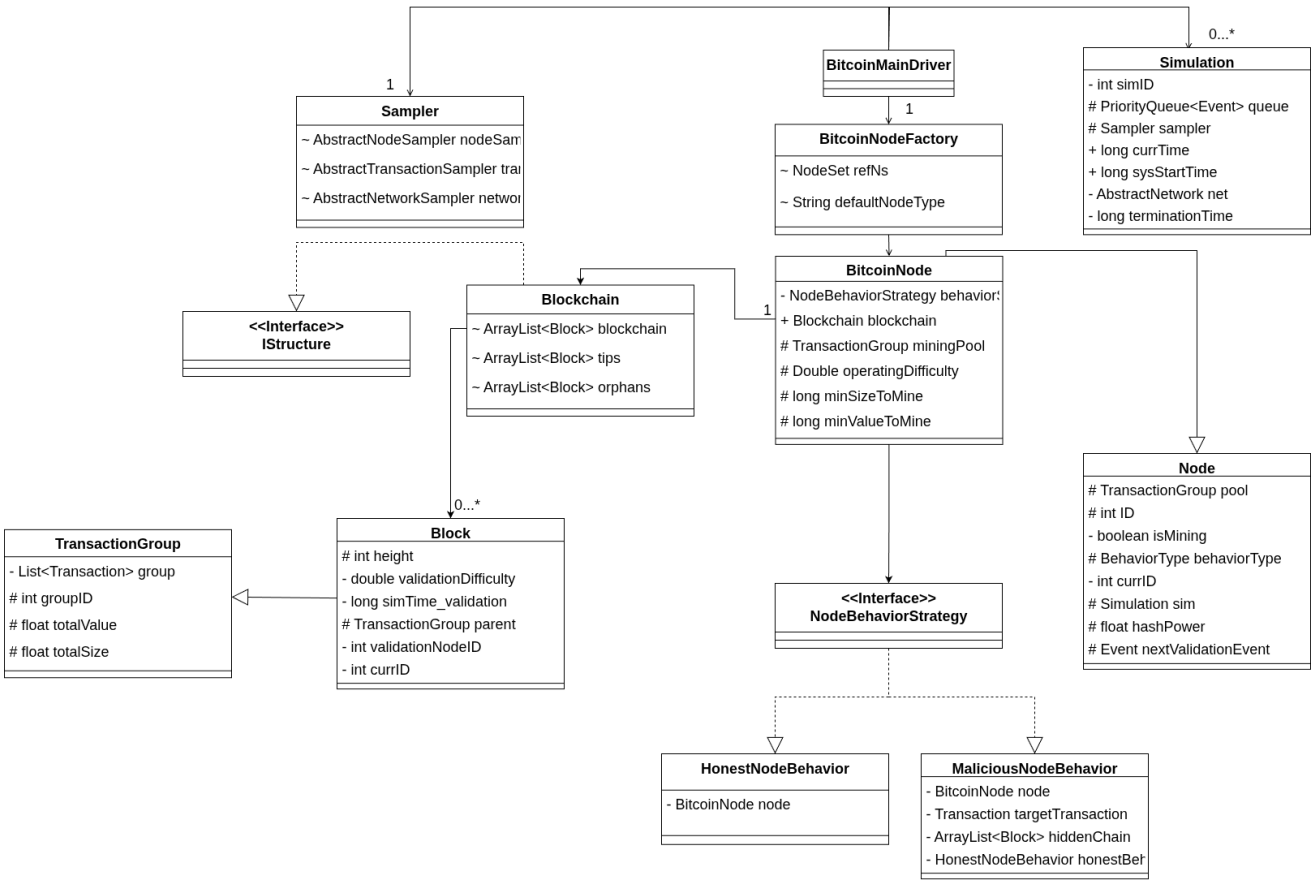

Figure 3.8: Hierarchical and modular design of the Bitcoin module and its interactions with associated classes.

3.3.1 Specialized Node Behavior in Bitcoin

The Bitcoin simulation builds on the generic node abstraction by implementing a `BitcoinNode` class that encapsulates protocol-specific operations. Key aspects of Bitcoin node behavior include:

- **Mining Operations:** Each Bitcoin node maintains a local mining pool of pending transactions and continuously ie., as part of the node's response to any event assesses whether the accumulated transaction value meets the threshold for mining. Container validation events (see 1), which signify successful conclusion of a mining run – a series of trials to identify a winning nonce – (see Section 2.2.2) are scheduled probabilistically based on expected values for the number of trials (hashes) needed for the node to succeed (determined by network difficulty) and for the time the node would need to carry out those trials (determined by its hash power).

- **Transaction Handling:** Bitcoin nodes handle transactions through a series of coordinated events that manage both locally arriving transactions and those received from the network. The process begins with the `Event_NewTransactionArrival` event, which triggers when a transaction is first received, either from a local client or the network. Valid transactions are then added to the node's transaction pool and scheduled for propagation through the `Event_TransactionPropagation` event. The transaction pool is continuously monitored and updated as new transactions arrive and existing ones are included in blocks. When a block is received through the `Event_ContainerArrival` event, nodes update their transaction pools by removing transactions that have been

confirmed in the new block. This event-driven transaction handling system ensures network integrity while maintaining efficient propagation and pool management, ultimately triggering mining events when the pool contains sufficient transactions to form a new block.

- **Behavioral Strategies and Adaptability:** Nodes can be configured to follow various strategies ranging from strictly honest operations to more aggressive tactics such as performing majority attack. These are implemented through application of the strategy design pattern. Specifically, each bitcoinNode object maintains associations with interchangeable objects that implement specific strategies for reacting to incoming events. The node delegates key behavioral decisions to those behavior objects, in a way that replacing a behavior object with one of a different type alters the behavior of the node. The `BitcoinNodeFactory` is responsible for instantiating nodes with the desired properties and behaviors, ensuring a heterogeneous network that reflects diverse real-world conditions. As discussed below, applications of the Strategy and Abstract Factory patterns to solve the variability handling problem of the previous version of CNSim is one of the key contributions of this thesis.

3.3.2 Blockchain and Block Management

A robust blockchain model is fundamental to simulating Bitcoin accurately. In CNSim, this is achieved through dedicated blockchain management components:

- **Block Creation and Validation:** The `Block` class, which extends from a generic

transaction group, captures essential block attributes such as height, timestamp, parent block reference, and the originating node's ID. When a node mines a new block, it undergoes a validation process governed by the `Blockchain` class. This validation ensures that each block adheres to the consensus rules, including proper parent-child relationships and non-overlapping transaction sets.

- **Orphan Block Resolution:** In scenarios where a block is received without its corresponding parent, the block is temporarily classified as an orphan. The simulator later attempts to reconcile orphan blocks once the missing parent block is integrated into the chain, thereby preserving the continuity of the blockchain.

3.3.3 Integration with the Core Simulation Engine

The Bitcoin module leverages CNSim's efficient event-driven engine to synchronize protocol-specific operations:

- **Synchronized Event Processing:** All Bitcoin-related events such as mining attempts, transaction broadcasts, and block validations are enqueued in the simulation's priority queue. This guarantees that events occur in a strictly chronological order, faithfully mirroring the temporal aspects of the Bitcoin network.

- **Customizable Simulation Scenarios:** Detailed configuration parameters allow researchers to tailor mining difficulty, network latency, transaction arrival rates, and node behaviors. This flexibility facilitates the exploration of a wide range of scenarios,

from normal network operations to stress-testing under adversarial conditions.

3.3.4 Reporting and Data Analysis

Comprehensive reporting is integrated into the Bitcoin simulation module to support post-simulation analysis:

- **Mining and Block Statistics:** The simulation tracks key metrics such as block mining rates, propagation delays, and chain reorganizations, providing insights into the efficiency and security of the mining process.
- **Transaction Throughput and Validation Times:** Detailed logs capture the performance of transaction processing, allowing researchers to assess how quickly and accurately transactions are validated and propagated across the network.
- **Node Performance Metrics:** The performance of individual nodes including the impact of various behavioral strategies is monitored and recorded, enabling a deeper understanding of their roles in maintaining network stability.

3.3.5 Configuration and Parameterization

The behavior of both the core simulation engine and the Bitcoin-specific module in CNSim is governed by a property-based configuration file. This file centralizes parameters such as simulation duration, reporting intervals, node composition, and protocol-level constraints (e.g., block size). Researchers can systematically adjust these parameters to explore a wide

range of scenarios without modifying the underlying codebase.

Broadly, the configuration file is divided into sections:

- **Simulation-wide Parameters:** Control global aspects like the total number of simulations (`sim.numSimulations`), the maximum number of nodes (`sim.maxNodes`), and when to terminate the simulation (`sim.terminate.atTime`).
- **Reporter Parameters:** Determine the granularity of output (e.g., whether to log individual transactions, node status, or detailed network events) through properties like `reporter.reportEvents` or `reporter.reportTransactions`.
- **Network and Workload Parameters:** Specify how nodes connect to each other and how transactions are generated. Examples include throughput means (`net.throughputMean`), propagation times (`net.propagationTime`), transaction arrival rates (`workload.lambda`), and the total number of transactions (`workload.numTransactions`).
- **Node Parameters:** Influence individual node behavior, such as electric power costs, malicious power ratios, or whether nodes can be adversarial. This section allows the simulation of diverse real-world conditions (e.g., mixed honest and malicious populations).
- **Hashpower Parameters:** Control Proof-of-Work difficulty (`pow.difficulty`) and the statistical distribution of computing power across nodes (`pow.hashPowerMean`, `pow.hashPowerSD`), enabling experiments under various mining conditions.

- **Bitcoin-Specific Parameters:** Configure properties like `bitcoin.maxBlockSize` and `bitcoin.minValueToMine`, dictating the rules for block creation, transaction inclusion, and other protocol details.

- **Bitcoin Reporter Parameters:** Further refine the reporting scope to capture block-level events or blockchain structure changes. This allows targeted tracking of the consensus process and network reorganizations.

When CNSim starts, it loads and applies all specified parameters ranging from general simulation settings to Bitcoin-specific rules thereby creating a consistent environment for repeatable and fine-grained experimentation. This modular configuration approach empowers researchers to systematically adjust factors like network latency, mining difficulty, and node composition while maintaining a single, centralized setup file for each study.

3.3.6 Summary

The Bitcoin network simulation module within CNSim extends the generic simulation framework to address the unique challenges of modeling Bitcoin. By integrating specialized node behaviors, rigorous blockchain management, and synchronized event scheduling, CNSim provides a realistic and scalable environment for studying Bitcoin's protocol dynamics. This module not only enables detailed performance and security analyses but also serves as a foundation for experimenting with alternative blockchain models and advanced distributed consensus mechanisms.

3.4 Implementing the Majority attack in CNSim's Bitcoin module

3.4.1 Introduction of Malicious Node Behavior

A significant contribution of this thesis to the CNSim framework is the introduction of a new node behavior designed to execute the Majority Attack on Bitcoin [1, 13]. Prior to this, such malicious behavior was not supported by the existing framework. To incorporate this attack, the Bitcoin node structure was modified to allow for dynamic behavior switching. As discussed in the following sections, this was achieved by integrating the Strategy design pattern into the node architecture, enabling nodes to alternate between honest and malicious behaviors during the simulation. This implementation builds upon previous work, such as the Majority Attack simulations in [51], where similar malicious behavior were introduced in other blockchain simulators.

3.4.2 Behavior Strategy Pattern

A key aspect of the modifications in this thesis is the introduction of the Behavior Strategy Pattern, which was not present in the original CNSim architecture. In the previous implementation, Bitcoin nodes were constrained to a single, fixed behavior (honest). By refactoring the node design, I introduced a mechanism that allows for the dynamic switching of node behaviors, in line with the Strategy Pattern described by Gamma et al. [52].

This modification involves defining the NodeBehaviorStrategy interface, along with corre-

sponding implementations for both honest and malicious behaviors. Nodes can now switch between these behaviors by altering the implementation of the NodeBehaviorStrategy interface they use, without needing to alter the node's core logic. The malicious behavior, specifically the Majority Attack, is encapsulated in a new implementation of this interface: MaliciousNodeBehavior. This class is responsible for managing the construction of a hidden chain and executing the attack at the appropriate moment.

The `BitcoinNode` class was modified to include the ability to switch between behaviors, allowing for both honest and malicious nodes, via coding to the `NodeBehaviorStrategy` interface and being agnostic to the actual type of the interface-implementing object it actually works with. The malicious behavior, encapsulated in `NodeBehaviorStrategy`-implementing `MaliciousNodeBehavior`, implements the Majority Attack. In this attack, a hidden chain is constructed and revealed at a strategic moment to double-spend a specific transaction. [1, 13]

3.4.3 Majority Attack Overview

The Majority Attack is a well-known vulnerability in the Bitcoin protocol, where an attacker controlling more than 50% of the network's mining power can rewrite the blockchain's history [1, 14]. By building a private chain and revealing it at a strategic moment, the attacker can execute a "double-spend" by reversing a transaction that the rest of the network already considers confirmed. This attack exploits Bitcoin's longest-chain rule, which dictates that honest nodes will always adopt the longest valid chain they are aware of.

The Attack Mechanism

The primary goal of the attack simulated in this thesis is to eliminate a transaction from the longest chain. The process, as implemented, unfolds in several distinct phases, triggered reactively.

Phase 1: Attack Initialization (The Trigger). The attack does not begin when the transaction is broadcast. Instead, the attacker waits until the specific `target transaction` is successfully mined into a block by any node on the network (either an honest miner or the attacker themself). The appearance of this block—the "trigger block"—marks the official start of the attack.

Phase 2: Private Fork and Secret Mining. Upon identifying the trigger block, the attacker immediately begins executing their malicious strategy. They start mining a new, private version of the blockchain (the "hidden chain"). This hidden chain is a **private fork** that begins from the *parent* of the trigger block. Crucially, the attacker ensures their hidden chain **omits** the target transaction. To do this, the attacker's node first removes the target transaction from its own local transaction pool so it will not be included in any new blocks they mine.

Phase 3: The Mining Race. While the honest network continues to build upon the trigger block, the attacker uses their majority hash power to grow their hidden chain in secret. Because the attacker has superior mining power, their hidden chain is expected to grow faster

and eventually become longer than the public chain.

Phase 4: The Reveal and Reorganization. Once the attacker's hidden chain is definitively longer than the public chain (and other strategic conditions are met), the attacker broadcasts their blocks to the network. Honest nodes, adhering to the longest-chain rule, are forced to perform a chain reorganization. They abandon the trigger block and all its descendants, adopting the attacker's longer chain as the new authoritative history. This action erases the target transaction from the ledger, completing the attack.

Attack Implementation and Parameters

This attack mechanism is implemented in CNSim through a specialized node behavior. The process is governed by several key parameters that define the attacker's strategy and the conditions for revealing the hidden chain.

Key Configurable Parameters. The simulation allows for the configuration of:

- The attacker's hash power, either as an absolute value or as a ratio relative to the honest network.
- The specific transaction to be targeted for the double-spend.
- Thresholds that control the timing of the attack reveal.

Chain Reveal Logic. In the simulation, the hidden chain is revealed when a specific set of conditions, defined by configurable parameters, is met. The attacker monitors the growth

of the public chain after the trigger block was mined and reveals their hidden chain when:

1. The hidden chain is longer than the public chain, ensuring it will be adopted by the network.

2. The number of blocks added to the public chain since the attack began is between a **Minimum Chain Length** (e.g., 6, to ensure the victim has accepted the transaction) and a **Maximum Chain Length** (e.g., 15, to act as a timeout).

This logic is expressed by the condition: ‘(hiddenChain.length > publicChain.length AND publicChainGrowth > MIN_LENGTH) OR publicChainGrowth > MAX_LENGTH‘.

Pseudocode Illustration

The core logic of the attack is executed through several functions that manage the attack’s lifecycle. The following pseudocode illustrates the process.

Algorithm 1 StartAttack Procedure

```
1: procedure StartAttack(Block b)
2:     Set isAttackInProgress to true
3:     Set blockchainSizeAtAttackStart to CalculateInitialBlockchainSize()
4:     LogAttackStart(b)
5: end procedure
```

Algorithm 2 HandleBlockValidation Procedure

```
1: procedure HandleBlockValidation(Block b, Time t)
2:     if isAttackInProgress then
3:         ValidateBlock(b)
4:         if Block is not in public blockchain then
5:             AddBlockToHiddenChain(b)
6:             ManageMiningAfterValidation()
7:             CheckAndRevealHiddenChain(b)
8:         end if
9:     end if
10: end procedure
```

Algorithm 3 CheckAndRevealHiddenChain Procedure

```
1: procedure CheckAndRevealHiddenChain(Block b)
2:     publicChainGrowth = CalculateChainGrowth()
3:     if ShouldRevealHiddenChain() then
4:         RevealHiddenChain()
5:     end if
6: end procedure
```

Algorithm 4 RevealHiddenChain Procedure

```
1: procedure RevealHiddenChain
2:     for each block b in hiddenChain (in reverse order) do
3:         Set Parent of b to previous block
4:         Add b to public blockchain
5:         Propagate b to network
6:     end for
7: end procedure
```

3.4.4 Implementation of the Attack in Simulation

The class `MaliciousNodeBehavior` was developed to handle the entire lifecycle of the Majority Attack in the simulation. This class is responsible for:

- Managing the construction and growth of the hidden chain, which forks from the parent of the trigger block.
- Monitoring the public chain's growth.
- Revealing the hidden chain at the appropriate time according to the defined logic.

The `MaliciousNodeBehavior` class extends the original honest behavior and incorporates the attack logic only when the attack is active. This allows the node to function as an honest participant during regular operations, switching to malicious behavior only when the attack is triggered.

Chapter 4

Experimental Study

4.1 Introduction and Research Questions

This chapter presents the experimental study conducted using CNSim to evaluate Bitcoin's transaction finality under various attack, network impairment, and misconfiguration scenarios. Each experimental scenario is designed to answer the research questions posed in the introduction:

1. **RQ1:** Can CNSim effectively simulate the Bitcoin network and perform finality-based evaluations of its consensus mechanism under realistic conditions?

2. **RQ2:** How does Bitcoin's transaction finality vary under Majority attack intensities (malicious power ratios), and how do these findings compare with existing literature on blockchain security?

3. **RQ3:** What is the impact of changing key network and protocol parameters (e.g., aver-

age end-to-end throughput, difficulty, block size, transaction arrival rate) on Bitcoin's finality, and how do these results align with or differ from current research findings?

The main motivation behind the above research questions is to assess the feasibility and practicality of the proposed finality-based approach to evaluating consensus quality, using Bitcoin and factors that affect its performance and security as a case study.

The chapter is structured as follows. First, we present experiments simulating a baseline Bitcoin-like network to assess CNSim's fidelity and confirm its ability to replicate known behaviors in a near-realistic scenario (addressing RQ1). Next, we introduce malicious actors and vary their power to understand the effect on consensus quality (RQ2). We then explore how changes in network and protocol parameters, such as difficulty, transaction arrival rate, and block size, impact finality (RQ3). We use these outcomes to reflect on the applicability of the finality-based approach to simulating consensus networks in the next and final chapter.

4.2 Definitions and Methodology

4.2.1 Baseline Bitcoin-Like Conditions

For **RQ1**, our objective is to configure CNSim with actual network parameters reflective of the Bitcoin mainnet. We input realistic parameters such as hashrate, difficulty, and block size, and then validate the simulation by measuring the emergent average block interval. If the resulting block production times are close to the observed 10-minute average from the real Bitcoin network, this constitutes evidence that our simulation environment successfully

replicates core Bitcoin dynamics. The data is driven from valid sources on 11 February 2025.

- **Mining Pools as Nodes:** We assign each of the major mining pools its real-world share of the total network hashrate, using data obtained from Blockchain.com for the 10 days prior to 11 Feb 2025 [53]. Table 4.1 shows an example snapshot of mining pool distribution, where each pool's share (%) corresponds to a proportionate amount of hashing power. By modeling each large pool as a distinct node (e.g., Unknown with 49.47%, AntPool with 16.78%, etc.), and potentially grouping smaller pools or individual miners, we create a diverse and realistic hashrate landscape within *CNSim*.

  Note that in our experiments, we did not consider the dynamic joining or leaving of nodes. Therefore, the community of witnesses remains the same and is considered active throughout each experimental run. In other words, all simulated nodes are continuously active.

Table 4.1: Bitcoin Mining Pool Distribution (Data from [53]).

Pool	**Share (%)**	**Blocks Mined**
Unknown	49.47	280
AntPool	16.78	95
ViaBTC	12.9	73
F2Pool	10.78	61
Mara Pool	4.42	25
Braiins Pool	1.59	9
SBI Crypto	1.41	8
BTC.com	1.24	7
BTC M4	0.88	5
Poolin	0.32	2
Ultimus	0.18	1

- **Total Hashrate and Difficulty:**

In *CNSim*, mining difficulty is defined as the ratio of the *search space* to the *success space*, directly mirroring Bitcoin's core design. Mathematically, the *search space* is the full range of valid hash outputs (2^{256} possible hashes), and the *success space* consists of all hashes below the network-defined target threshold. The mining difficulty D can thus be expressed as:

$$D = \frac{\text{Search Space}}{\text{Success Space}} = \frac{2^{256}}{\text{Target Threshold}}.$$

To ensure fidelity to Bitcoin, we employ a custom Java utility that calculates this ratio based on Bitcoin's `INITIAL_TARGET` (the threshold value corresponding to a difficulty of 1) and `MAX_HASH_VALUE` (the total 2^{256} search space). The utility derives the *CNSim* difficulty using the real-world Bitcoin difficulty. Given the current Bitcoin difficulty D_{BTC}, the equivalent *CNSim* difficulty D_{CNSim} is calculated as:

$$D_{\text{CNSim}} = D_{\text{BTC}} \cdot \text{CNSIM\_INITIAL\_DIFFICULTY},$$

where the scaling factor CNSIM_INITIAL_DIFFICULTY is defined as:

$$\text{CNSIM\_INITIAL\_DIFFICULTY} = \frac{1}{\frac{\texttt{INITIAL\_TARGET}}{\texttt{MAX\_HASH\_VALUE}}}.$$

This formula ensures that a Bitcoin difficulty of $D_{\text{BTC}} = 1$ maps to the correct initial CNSim difficulty. As an example, using real-world Bitcoin difficulty data (e.g., $D_{\text{BTC}} =$

114.17×10^{12} as per CoinWarz [54]), we obtain:

$$D_{\text{CNSim}} = 4.90364 \times 10^{23}.$$

By setting this difficulty in the simulation, we ensure that CNSim accurately reflects Bitcoin's network conditions. After running the simulation, we verify the average block generation interval. If this interval approximates Bitcoin's target of 10 minutes, it confirms that the translation of Bitcoin difficulty and total hashrate into *CNSim* correctly reproduces Bitcoin's block generation process.

- **Block Size:** We configure the block size limit to the standard Bitcoin block size of 1 MB, which corresponds to 1,000,000 bytes of raw transaction data.

- **Transaction Arrival Rate and Configuration:** To accurately simulate real-world Bitcoin network conditions, we configure the transaction arrival rate in *CNSim* based on empirical data. According to Blockchain.com [55], the transaction arrival rate for the observed date (10th February 2025) is $\lambda = 4.16$ transactions per second. This rate translates to a total number of transactions included in the workload as follows:

$$\text{Total Transactions} = 4.16\,\text{tx/sec} \times 18,000\,\text{sec} \approx 75,000\,\text{transactions}.$$

Transaction Fee Configuration: The average transaction fee, as obtained from Blockchain.com [55], is reported as $1.254 USD. With the Bitcoin-to-USD exchange

rate at the time being 1 BTC = \$95,893 USD, the equivalent fee in Bitcoin is calculated as:

$$\frac{1.254}{95,893} = 0.0000130728\,\text{BTC} = 1,307.28\,\text{satoshis}.$$

We assume a coefficient of variation (CV) of 10% for transaction fees, leading to a standard deviation of:

$$\sigma_{\text{fee}} = 0.1 \times 1,307.28 = 130.73\,\text{satoshis}.$$

Transaction Size Distribution: According to Bitcoin Visuals [56], the average transaction size is 460 bytes, but further analysis of the distribution percentiles suggests a long-tailed distribution. The 10th, 50th (median), and 90th percentiles are reported as follows:

- $p_{10} = 181$ bytes
- $p_{50} = 225$ bytes (median)
- $p_{90} = 491$ bytes

While the transaction size distribution appears skewed, we make the simplifying assumption of normality centered around the median (225 bytes). To determine the standard deviation, we use the 10th and 90th percentiles along with their corresponding quantiles from the standard normal distribution :

$$z_{10} = q_{\text{norm}}(0.1), \quad z_{90} = q_{\text{norm}}(0.9)$$

Using these quantiles, the standard deviation σ is estimated as:

$$\sigma_{\text{size}} = \frac{p_{90} - p_{10}}{z_{90} - z_{10}} = \frac{491 - 181}{1.2816 - (-1.2816)} = 120.9471 \text{ bytes.}$$

Final Workload Configuration: Based on the empirical data and assumptions stated above, the final parameters for transaction simulation in *CNSim* are as follows:

- **Transaction arrival rate:** $\lambda = 4.161$ transactions per second.
- **Total transactions:** 75,000 transactions.
- **Mean transaction size:** 225 bytes.
- **Transaction size standard deviation:** 120.9471 bytes.
- **Mean transaction fee:** 1,307.28 satoshis.
- **Transaction fee standard deviation:** 130.73 satoshis.

By integrating these real-world transaction characteristics, *CNSim* provides a more realistic yet computationally efficient simulation of Bitcoin's network behavior. This configuration allows for a balance between accuracy and feasibility, particularly for modeling block creation, transaction propagation, and consensus mechanisms.

4.2.2 Metrics and Measurement Methodology

To quantitatively evaluate transaction finality and overall network performance, we employ a set of specific metrics. These metrics are applied consistently across all experimental scenarios (RQ1, RQ2, and RQ3) to ensure comparability. Performance analyses are conducted on a large cohort of transactions (e.g., the first 25,000 or 50,000) to capture system-wide behavior rather than results from single transactions.

- *Transaction confidence score:* For a specific target transaction, we measure its confidence score over time. This score is defined as the percentage of network nodes that have accepted the block containing the transaction into their main chain at a specific time. A transaction is considered to have reached finality when this score reaches a high threshold (e.g., 90%) and remains stable. This metric is crucial for RQ2, where we plot the confidence for transaction ID 20000 to visualize the direct success or failure of a targeted attack.

- *Transaction finalization time:* This is the primary metric for measuring performance for a group of transactions. We define finalization time as the duration from a transaction's creation time to the moment it achieves finality, i.e., it is accepted by at least by a minimum proportion c of the active nodes – in our experiments it is set to 0.9. This state must be stable until the end of the simulation, as per finality definition.

- *Consensus progression curve:* To visualize the performance across the entire network, we introduce the *consensus progression curve*. This is a plot showing the fraction of

total transactions that have been finalized (as defined above) over time. Key data points from this curve include the median (50%) and 90th percentile finalization times.

- **Chain Stability Metrics:** In addition to the above novel visualizations and metrics we also use more established measures of consensus quality namely:
  - *Mean final chain height:* The average number of blocks in the longest chain at the end of the simulation. A lower height under the same time duration indicates a slower block production rate.
  - *Fork rate:* The average number of forked blocks per simulation run. A high fork rate indicates consensus instability, often caused by high network latency or malicious activity.

4.2.3 Methodology for Analyzing Malicious Power Ratio (RQ2)

To address RQ2, our methodology evaluates the impact of malicious hashrate on transaction finality from two distinct perspectives: a targeted analysis of a single transaction and a system-wide analysis of overall network performance.

Targeted Transaction Analysis First, to measure the direct effectiveness of the attack, we perform a *targeted transaction analysis*. This involves selecting a single, specific transaction (ID 20,000, arbitrarily chosen) as the adversary's target. We empirically estimate the theoretical finality metric (defined in Chapter 3) by tracking the transaction'sconfidence score over time. We also calculate its overall *finality success rate*, which we define as the percentage

of independent simulation trials (out of 30) in which the targeted transaction successfully reaches finality.

System-Wide Impact Analysis Second, to quantify the effect of the attack on the broader network, we conduct a *system-wide impact analysis*. For this, we analyze the finalization performance of a large cohort of transactions—specifically, the first 50,000 transactions introduced into the simulation. The primary tool for this analysis is the consensus progression curve, which visualizes systemic delays and disruptions to finality for non-targeted transactions.

To clarify the relationship between our analytical tasks and metrics, Table 4.2 provides a summary.

Table 4.2: Summary of Metrics and Visualizations for RQ2 Analysis.

Analysis Task	**Associated Metric**	**Visualization**	**Metric Definition**
Targeted Transaction Analysis	Finality Success Rate	Confidence Score Plot	**Finality Success Rate:** The percentage of simulation runs where the target transaction reaches finality (90% confidence).
System-Wide Impact Analysis	Transaction Finalization Time (Median, 90th percentile)	Consensus Progression Curve	**Finalization Time:** The time from transaction creation until it is stably accepted by 90% of network nodes.

4.2.4 Methodology for Analyzing Network and Protocol Variations (RQ3)

The methodology for addressing RQ3 is designed to isolate and quantify the impact of individual network and protocol parameters on system-wide transaction finality. The approach involves systematically adjusting a single parameter from the established baseline configuration while holding all others constant. The key parameters investigated are:

- **Mining Difficulty:** Varied to be higher or lower than the baseline to measure the effect on block discovery rates and finalization speed.
- **Network Throughput:** Adjusted to simulate different network conditions and measure the impact on block propagation, fork rates, and finality.
- **Block Size and Transaction Arrival Rate:** Modified to simulate varying degrees of network load and mempool congestion.

For each parameter variation, we perform a system-wide performance analysis by measuring the finalization times for a cohort of the first 25,000 transactions. This large sample size ensures that our findings reflect the overall system's response rather than anecdotal outcomes. The primary metric for this comparative analysis is the consensus progression curve. By overlaying the curve of each experimental scenario with that of the baseline, we can directly and quantitatively assess how each parameter change affects the speed and reliability of transaction finality across the network.

4.3 Summary of Experimental Parameters

To ensure a comprehensive analysis, we configure *CNSim* with realistic Bitcoin-like parameters for the baseline and explore variations across several dimensions. To verify the statistical significance of the results, we run 30 independent simulation trials for each configuration, using different random seeds. These configurations are detailed in the following tables.

Table 4.3: Key Configuration Parameters (Baseline and Variations)

Parameter	**Baseline (Bitcoin-like)**	**Variations**
Number of Nodes	11	N/A
Block Size	1 MB	0.5 MB, 2 MB
Malicious Power	0%	20%, 35%, 62%
Network Bandwidth	25 Mbps	1 - 100 Mbps
Hashrate Distribution	Real-world mining pool shares	Custom Shares
Total Hashrate	717.14 EH/s	717.14 EH/s
Transaction Arrival Rate	4.161 tx/sec	8.161 tx/sec, 2.161 tx/sec

Mining Pool Configuration

To emulate a realistic mining environment, *CNSim* is configured using the following mining pool distribution, based on data from Blockchain.com [53] at Block 874,145:

Simulation Parameters

We use a standard block size of 1 MB for the baseline configuration, consistent with Bitcoin's operational parameters. Each simulation run is designed to represent multiple hours of blockchain operation, ensuring the system achieves steady-state behavior.

To simulate almost five hours of operation at an arrival rate of 4.161 transactions per

Table 4.4: Bitcoin Mining Pool Distribution (Data from [53]).

Pool	**Share (%)**	**Blocks Mined**
Unknown	49.47	280
AntPool	16.78	95
ViaBTC	12.9	73
F2Pool	10.78	61
Mara Pool	4.42	25
Braiins Pool	1.59	9
SBI Crypto	1.41	8
BTC.com	1.24	7
BTC M4	0.88	5
Poolin	0.32	2
Ultimus	0.18	1

second, the workload includes:

$$5\,\text{hours} \times 3,600\,\text{sec/hour} \times 4.161\,\text{tx/sec} = 75,000\,\text{transactions}.$$

This workload reflects realistic transaction volumes, with each transaction characterized by an average size of 225 bytes and a standard deviation of 120.9471 bytes (calculated using the 10th and 90th percentiles). Average transaction fees are set to 1,307.28 SATS, with a standard deviation of 130.73 SATS (assuming a coefficient of variation of 10%).

Hardware Configuration and Runtime: The simulations are run on a modern machine with the following specifications:

- CPU: AMD Ryzen 5 7640U processor (6 cores, 12 threads, up to 4.97 GHz)
- RAM: 32 GB DDR4

Each simulation for the baseline configuration requires approximately 1 hour of real-time

to simulate 5 hours of blockchain activity. Variations in block size, malicious power, and transaction arrival rates are similarly tested, with runtimes adjusted proportionally.

4.4 Experiments and Results

4.4.1 RQ1: Baseline Bitcoin-Like Scenario

To formally validate the model's temporal dynamics, we performed a rigorous statistical comparison against real-world data. The validation uses two datasets: the **916 block intervals** generated from 30 simulation runs and the complete set of **4,294 real-world block intervals** from February 2024, sourced from Blockchair.com [57]. The descriptive statistics for both datasets are summarized in Table 4.5.

Table 4.5: Descriptive Statistics for Block Interval Data (minutes).

Statistic	**Simulated Data**	**Real-World Data**
Sample Size (n)	916	4,294
Mean ($\bar{x}$)	8.88	9.72
Std. Dev. (s)	8.65	9.39

A **Two One-Sided Tests (TOST)** procedure was used to determine if the difference between the means is small enough to be considered practically equivalent. The visual similarity of the two distributions is confirmed in the Empirical Cumulative Distribution Function (eCDF) plot in Figure 4.1.

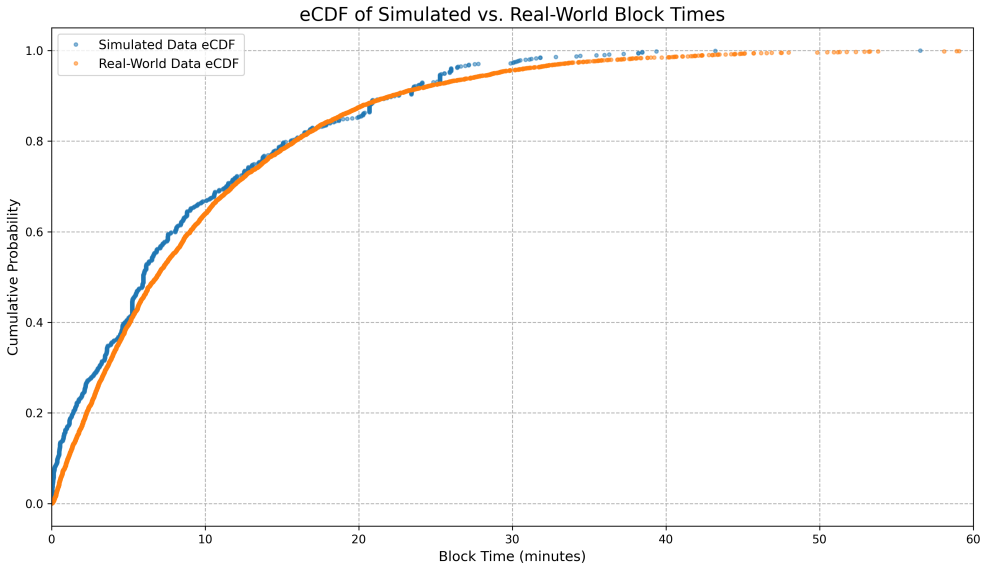

Figure 4.1: eCDF plot of simulated vs. real-world block times. The close alignment of the two lines provides strong visual evidence that the distributions are statistically similar.

TOST Procedure Calculation

1. **Define Equivalence Margin (Δ):** Based on the practical requirements of the model, an equivalence margin of $\Delta = \pm 1.5$ minutes was established.

2. **Calculate Pooled Standard Deviation (s_p):** The pooled standard deviation, which combines the variance of both samples, was calculated as:

$$s_p = \sqrt{\frac{(n_{sim} - 1)s_{sim}^2 + (n_{real} - 1)s_{real}^2}{n_{sim} + n_{real} - 2}} = 9.26 \text{ min}$$

3. **Calculate T-Statistics:** With a difference in means of -0.84 minutes and a standard error of 0.34 minutes, the two one-sided t-statistics were calculated against the bounds of the equivalence margin:

- Upper Bound Test: $t_1 = \frac{(\bar{x}_{sim} - \bar{x}_{real}) - \Delta}{SE_{diff}} = -6.95$
- Lower Bound Test: $t_2 = \frac{(\bar{x}_{sim} - \bar{x}_{real}) - (-\Delta)}{SE_{diff}} = 1.97$

4. **Determine Final P-Value:** The larger of the two p-values from these tests is taken as the final result. With 5,208 degrees of freedom, the resulting p-value is **0.0245**.

Since this p-value is less than our significance level of $\alpha = 0.05$, we reject the null hypothesis of non-equivalence. This result provides strong statistical evidence that the mean block time of our simulation is practically and statistically equivalent to the live Bitcoin network.

Overall Network Performance

With the model's core timing mechanism validated, we analyzed overall network performance. The baseline performance metrics, averaged over 30 runs, are summarized in Table 4.6. The network demonstrated high stability with zero forked blocks.

Table 4.6: Summary of Baseline Scenario Performance Metrics.

Metric	**Value**
Average Final Chain Height	30.53
Average Block Time (minutes)	9.17
Average Fork Rate (forks/run)	0
Median Transaction Finalization Time (minutes)	8.25
Time to Finalize 90% of Transactions (minutes)	13.42

Figure 4.2 provides a cumulative distribution of these finalization times for the first 25,000 transactions. As defined in our methodology, "finalization time" is the time from transaction creation until it is included in a block accepted by 90% of the network nodes, and remains

accepted till the end. The curve shows a rapid confirmation rate, with the data confirming that **50% of transactions (the median) reach finality within approximately 8.25 minutes**. Furthermore, **90% of transactions are finalized across the network in just over 13.4 minutes**. This behavior, where a supermajority of nodes agrees on a transaction's validity within one to two block intervals, aligns with empirical observations of the live Bitcoin network and validates CNSim's fidelity in modeling consensus dynamics.

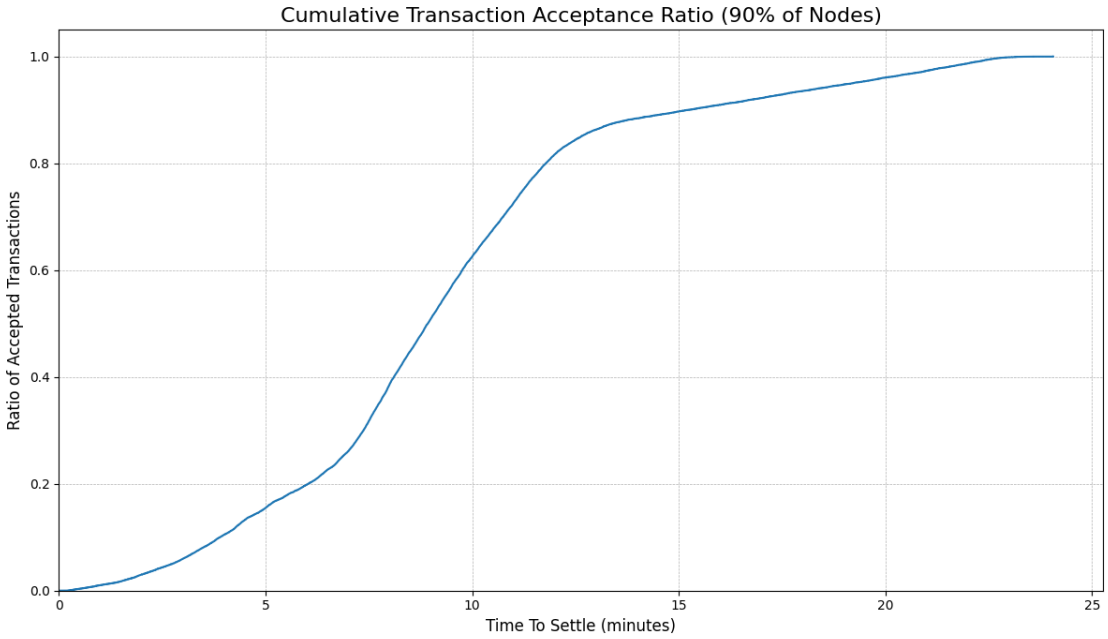

Figure 4.2: Consensus progression curve showing the cumulative distribution of transaction finalization time for the baseline scenario. The y-axis shows the fraction of transactions accepted by at least 90% of nodes, and the x-axis shows the time to reach this threshold in minutes.

4.4.2 RQ2: Analyzing Network Resilience to Malicious Power

To assess the robustness of the finality score under adversarial conditions, we simulated scenarios where mining pools engage in malicious behavior. The specific strategy is a targeted

attack on a single transaction (ID 20000). In this attack, the malicious actor mines a block containing the target transaction, broadcasts it, and then immediately attempts to build a longer, hidden chain that excludes that block, aiming to force a chain reorganization and invalidate the target's initial confirmation.

To obtain a more accurate conclusion about the systemic impact of this attack, our aggregate performance analysis evaluates the finalization of the first **50,000 transactions**. This expanded scope allows us to observe the effects not only on the targeted transaction but also on the thousands of transactions that join the network while the attack is ongoing.

The following scenarios are considered, all targeting the block containing transaction ID 20000:

- **Scenario 1**: A minority attack where AntPool, controlling 16.78% of the network hashrate, acts maliciously.
- **Scenario 2**: A significant minority attack where AntPool (16.78%) and ViaBTC (12.9%) cooperate, resulting in 29.68% malicious power.
- **Scenario 3**: A majority attack where a coalition of "Unknown" miners (49.47%) and AntPool (16.78%) control 66.25% of the network hashrate.

Scenario 1: 16.78% Malicious Power

With 16.78% of the hashrate, the malicious actor's influence was negligible on the target transaction. The mean chain height saw a minor drop from a baseline of **30.53** to **28.70**. This drop is attributed to the malicious actor attempting to build a separate, hidden chain rather than contributing to the main chain. The attack was ultimately unsuccessful, as the honest network consistently outpaced the attacker, and the blocks from the malicious hidden chain were discarded as stale blocks.[2] (Table 4.7). The attack was completely ineffective against its intended target; the time to finality for transaction ID 20000 was **21.62 minutes**, nearly identical to the baseline of 21.63 minutes. The transaction successfully reached finality in 100% of the simulation trials (Table 4.8). This finding confirms the analytical models of Aponte-Novoa et al. [58], which show that the probability of a successful double-spending attack is extremely low when the attacker possesses a small fraction of the network's hash rate.

Contrary to the intuition that a failed attack on a single transaction would be harmless to the wider network, our results show this minor attack slightly degraded overall network performance. The time required to finalize 90% of all transactions *increased* from **12.74 minutes** in the baseline to **16.36 minutes** (Table 4.9). This indicates that even a failed, low-power attack introduces enough disruption to slow down the finalization of non-targeted transactions across the network; however, it does not have enough power to make an impact

[2]In blockchain terminology, a **stale block** is a valid block that is solved correctly but is not included in the final, canonical chain because a competing block at the same height was accepted and extended by the network first. In contrast, an **orphan block** is a valid block whose parent block has not yet been received by a node; it is held in memory until its parent arrives, at which point it can be connected.

on the specific target transaction. This outcome aligns with findings by Zhu et al. [59] that stubborn mining attacks can "extremely downgrade the system throughput," and our consensus progression curve provides a finality-based visualization of this same performance degradation.

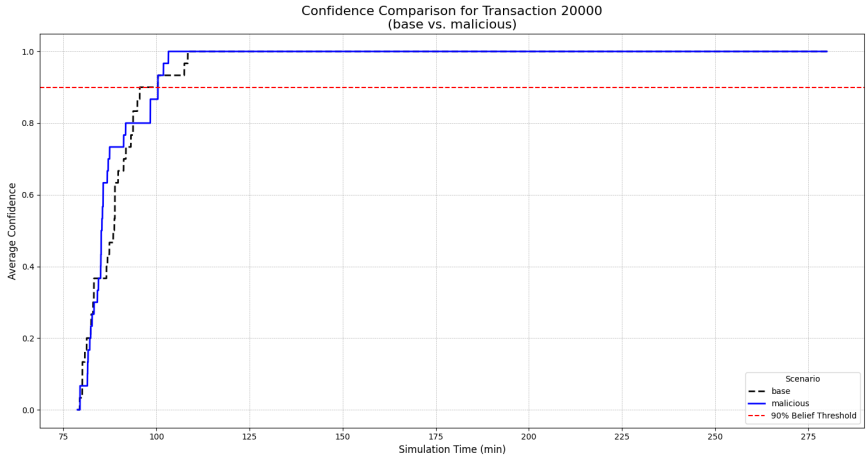

Figure 4.3: Average Confidence of the targeted transaction (ID 20000) under a 16.78% attack over 30 experimental runs.

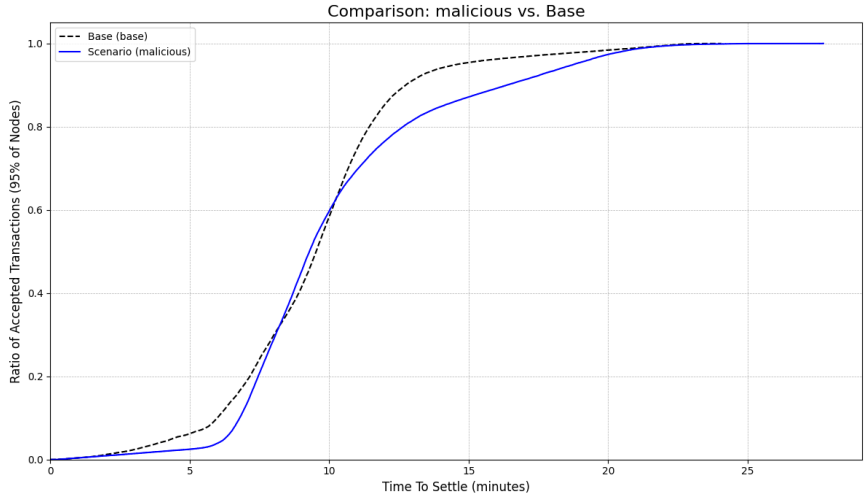

Figure 4.4: Consensus progression curve for the first 50,000 transactions under a 16.78% attack.

Scenario 2: 29.68% Malicious Power

As the malicious power increased to 29.68%, the impact on the network became significant. The mean chain height degraded further to **25.10**, and the network began experiencing forks, with an average of **0.6** orphaned blocks per simulation (Table 4.7).

The most critical finding for this scenario is the introduction of **probabilistic failure** for the targeted transaction. The attack was powerful enough to prevent transaction ID 20000 from reaching finality in 6 out of 30 trials, resulting in a finality success rate of only **80%** (Table 4.8). This demonstrates that an attacker with just under 30% hashrate can make the confirmation of a targeted transaction unreliable. This finding can be compared to the analytical model used by Aponte-Novoa et al. [58], which calculated that a miner with 22% hash power could achieve a double-spend success probability of up to 55% against a single confirmation. Our simulation's finding of a 20% attack success rate highlights the practical challenges an attacker faces compared to simplified theoretical models. Furthermore, our results empirically identify a "benefit threshold"—a concept also explored by Zhu et al. [59] —where the attack strategy becomes credibly threatening. The attack also had a clear systemic impact, increasing the time to finalize 90% of all transactions from a baseline of **12.74 minutes** to **19.49 minutes**.

Scenario 3: 66.25% Malicious Power (Majority Attack)

With a dominant 66.25% of the network hashrate, the malicious coalition executed a devastatingly successful attack. This provides empirical validation of the theoretical definition

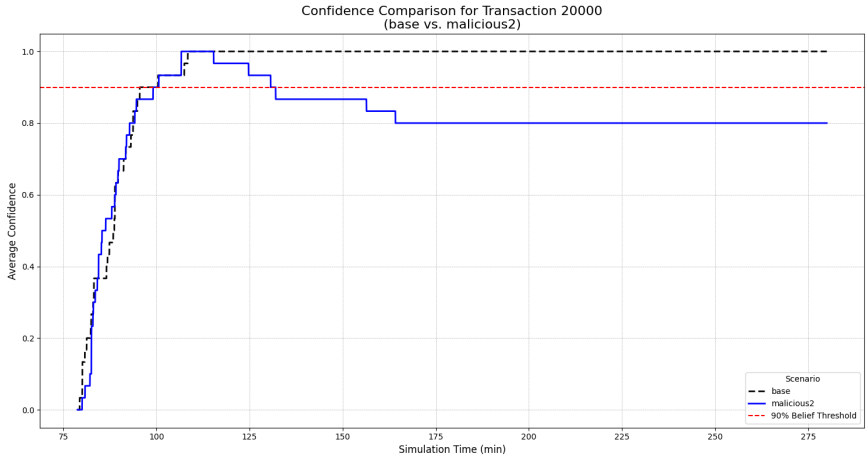

Figure 4.5: Average Confidence of the targeted transaction (ID 20000) under a 29.68% attack over 30 experimental runs.

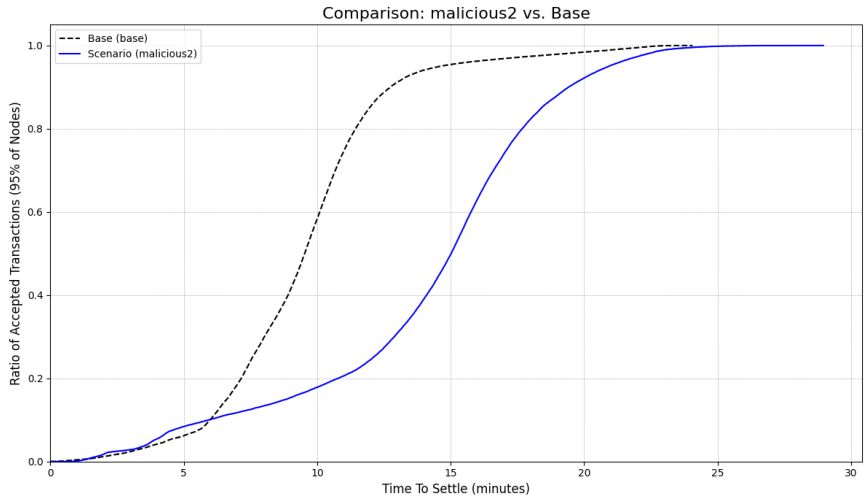

Figure 4.6: Consensus progression curve for the first 50,000 transactions under a 29.68% attack.

of the 51% attack,, as detailed by researchers like Aponte-Novoa et al. [58] and Zhou et al. [60], where an attacker with over half the computing power is capable of unilaterally altering the blockchain. The high number of forks (**3.47 on average**) reflects the constant chain reorganizations initiated by the attacker. The attack on transaction ID 20000 was always successful: it **failed to reach finality in 100%** of the trials (Table 4.8).

The systemic impact was equally catastrophic. The time to finalize 90% of transactions surged from **12.74 minutes** to **59.63 minutes**, more than quadrupling the baseline. Critically, the network failed to finalize 100% of transactions within the simulation window (Table 4.9), demonstrating that a majority attacker can effectively halt the network's ability to provide finalization guarantees for all participants.

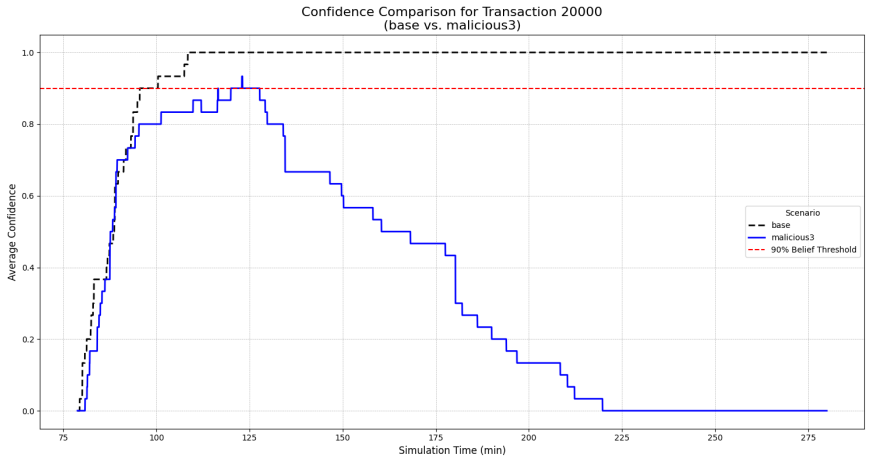

Figure 4.7: Average Confidence of the targeted transaction (ID 20000) under a 66.25% attack over 30 experimental runs.

Quantitative Summary and Conclusion

The progression across the three scenarios clearly illustrates the degradation of network health as malicious power increases. This aligns with the foundational principle established in research by Aponte-Novoa et al. [58] and Zhu et al. [59], which states that the probability of a successful attack is directly proportional to the attacker's share of the total network hash rate. Table 4.7 summarizes the impact on chain stability, while Table 4.8 details the starkly different outcomes for the targeted transaction. Table 4.9 shows the severe systemic impact

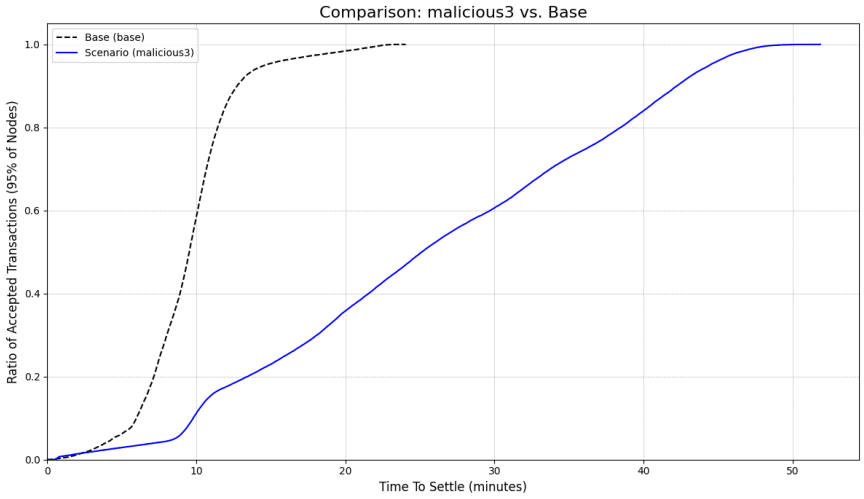

Figure 4.8: Consensus progression curve for the first 50,000 transactions under a 66.25% attack.

on aggregate finalization times.

Table 4.7: Chain stability metrics as malicious power increases.

Simulation Scenario	**Mean Height**	**Mean Fork Rate (forks/run)**
Baseline (No Attack)	30.53	0
Scenario 1 (16.78% Malicious)	28.70	0
Scenario 2 (29.68% Malicious)	25.10	0.60
Scenario 3 (66.25% Malicious)	29.80	3.47

Table 4.8: Finality metrics for the targeted transaction (ID 20000) under malicious attacks.

Scenario	**Finality Success Rate**	**Avg. Time to Finality (min)**
Baseline	100%	21.63
Scenario 1 (16.78%)	100%	21.62
Scenario 2 (29.68%)	80%	N/A
Scenario 3 (66.25%)	0%	N/A

In conclusion, these simulations validate that network finality degrades severely as malicious hashrate increases. Our new analysis reveals critical thresholds: a minority attack of **16.78%** is insufficient to compromise a specific target but still harms overall network

Table 4.9: Aggregate transaction finalization time (in minutes) under malicious attacks.

Scenario	Time to Finalize 25%	Time to 50%	Time to Finalize 75%	Time to Finalize 90%	Time to Finalize 100%
Baseline	**7.57**	**9.53**	**11.02**	**12.74**	**24.06**
Scenario 1 (16.78%)	7.77	9.29	11.74	16.36	27.71
Scenario 2 (29.68%)	12.15	15.27	17.48	19.49	28.98
Scenario 3 (66.25%)	15.76	24.22	42.88	59.63	N/A

latency. A significant minority power of **29.68%** is sufficient to make transaction finality unreliable, succeeding in its attack 20% of the time while significantly slowing down the rest of the network. Finally, a majority attack of **66.25%** makes targeted reversals trivial and cripples the network's ability to finalize transactions system-wide.

These results also highlight the effectiveness of our measurement methodology. To assess the outcome of a targeted attack, we introduced the finality success rate, defined as the percentage of simulation trials (out of 30) in which the target transaction successfully reached finality. This metric provides a clear, quantitative score of an attack's overall effectiveness (e.g., 100%, 80%, and 0% across our scenarios). The Confidence graphs (e.g., Figure 4.5) complemented this by providing an unambiguous, real-time visualization of how an attack succeeded or failed, showing confidence plummeting to zero or failing to stabilize. Simultaneously, the consensus progression curves (e.g., Figure 4.6) effectively captured the systemic "collateral damage" of the attack on non-targeted transactions, a crucial aspect that single-transaction analysis would miss. Together, these tools offer a multi-faceted view of network resilience, diagnosing both specific vulnerabilities and overall system degradation.

4.4.3 RQ3: Impact of Network and Protocol Parameter Changes

To address RQ3, we explore how varying key parameters affects transaction finalization time across the network. This analysis focuses on changes to mining difficulty, block size, network throughput, and transaction arrival rates. In this context, we define "finalization time" as the time elapsed from a transaction's first appearance in the network until it has been recorded by 90% of the nodes, and remaining till the end.

The subsequent sections analyze the findings for each parameter in detail, with a final summary table comparing all scenarios at the end.

The Impact of Mining Difficulty

Mining difficulty directly controls the rate of block discovery. Increasing difficulty leads to longer block times, while decreasing it has the opposite effect. For this analysis, all other parameters were held at their baseline values as described in Section 4.2.1: a transaction arrival rate of 4.161 tx/sec, a block size of 1 MB, a network bandwidth of 25 Mbps, and 11 nodes with hashrate distributed according to Table 4.1. The performance impact is summarized in Table 4.10.

Table 4.10: Performance Metrics for Mining Difficulty Variations.

Scenario	**Avg. Final Chain Height**	**Median finalization Time (min)**	**90% finalization Time (min)**
Baseline	**30.5**	**8.25**	**13.42**
Increased Difficulty	22.0	15.44	24.52
Decreased Difficulty	57.0	4.66	6.74

Quantitative Impact on Finalization The impact of changing mining difficulty aligns with theoretical expectations. Increasing the difficulty significantly slows down transaction finalization, with the median time to finalize increasing from **8.25 minutes** in the baseline to **15.44 minutes**. Conversely, decreasing the difficulty more than halved the median finalization time to just **4.66 minutes**. This is a direct consequence of the block discovery rate: as shown in Table 4.10, the low-difficulty scenario produced an average of **57 blocks**, nearly double the baseline, effectively increasing network capacity and speeding up confirmations.

However, this approach has a critical trade-off that limits its viability. As block discovery times shorten, they can become comparable to block propagation times, increasing the probability of forks and threatening consensus stability. While we did not observe any forked blocks in this specific scenario, a more aggressive difficulty reduction, or a larger time-window of simulation, would inevitably lead to network instability. This finding is consistent with research by Bistarelli et al. [61] and Cao et al. [62], who confirm that decreasing mining difficulty raises the probability of forks and that the negative impact of network latency is amplified when block intervals are short.

The Impact of Maximum Block Size

Block size limits the number of transactions per block, creating a trade-off between network throughput and block propagation time. Prior work by Shahsavari et al. [63] has established that while larger blocks can increase throughput, they also cause a significant increase in block propagation time. The results are shown in Table 4.11.

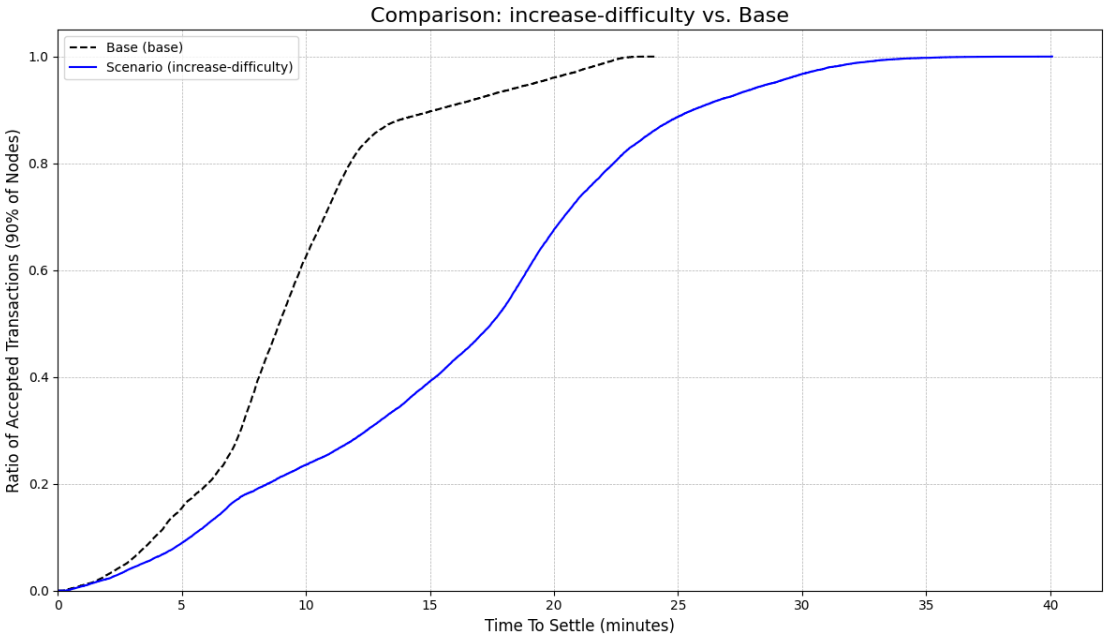

Figure 4.9: Comparison of finalization time with increased mining difficulty vs. the baseline.

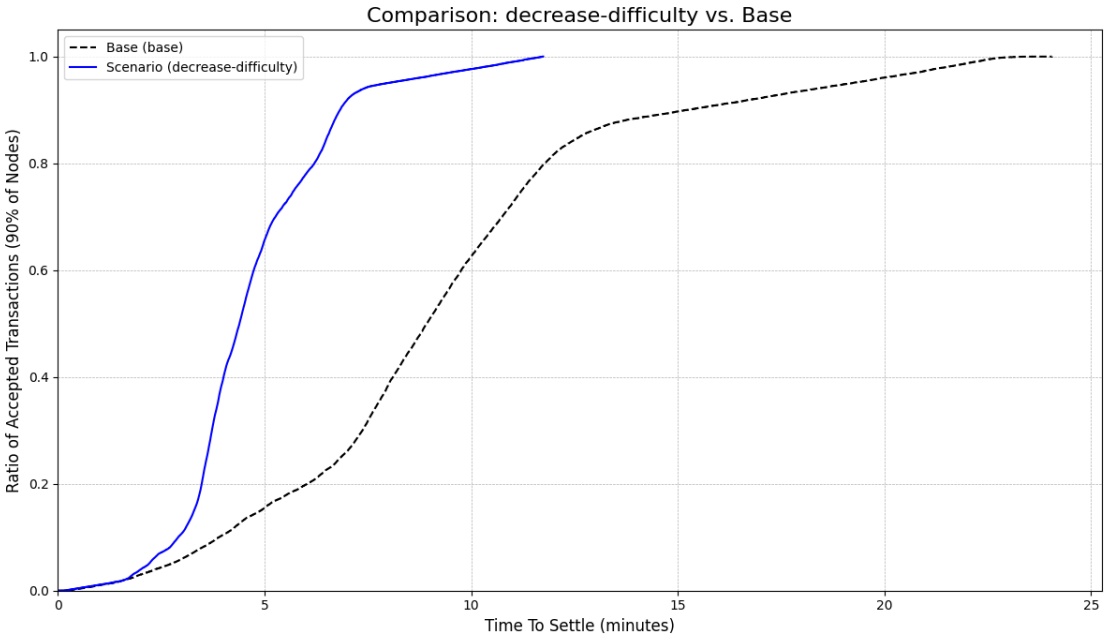

Figure 4.10: Comparison of finalization time with decreased mining difficulty vs. the baseline.

Table 4.11: Performance Metrics for Block Size Variations.

Scenario	Median finalization Time (min)	90% finalization Time (min)
Baseline (1 MB)	**8.25**	**13.42**
Doubled Size (2 MB)	7.97	12.84
Halved Size (0.5 MB)	12.45	23.98

Quantitative Impact on Finalization Halving the block size to 0.5 MB created a significant bottleneck. The time required to finalize 90% of transactions increased from **13.42 minutes** in the baseline to **23.98 minutes**. In contrast, doubling the block size to 2 MB yielded a modest improvement, with the 90% finalization time decreasing slightly to **12.84 minutes**.

The limited impact of doubling the block size is explained by the network's baseline workload. With an arrival rate of 4.161 tx/sec and an average transaction size of 225 bytes, the average data generated per 10-minute block interval is approximately 0.60 MB. This means the baseline 1 MB blocks were only about 60% full on average. Since the system was not constrained by block space, providing additional capacity with 2 MB blocks did not remove a bottleneck and thus offered no significant performance gain. A substantial improvement from larger blocks would only manifest under a much higher transaction arrival rate.

The figures below show that while the doubled block size offers a small advantage, the halved block size imposes a severe performance penalty.

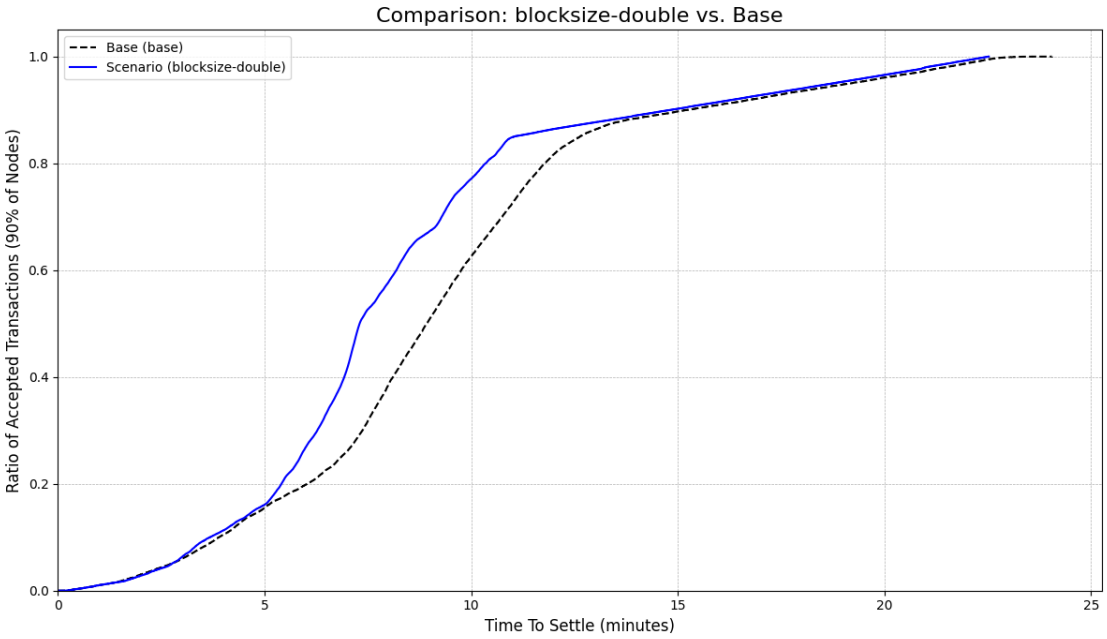

Figure 4.11: Comparison of finalization time with a doubled block size (2 MB) vs. the baseline.

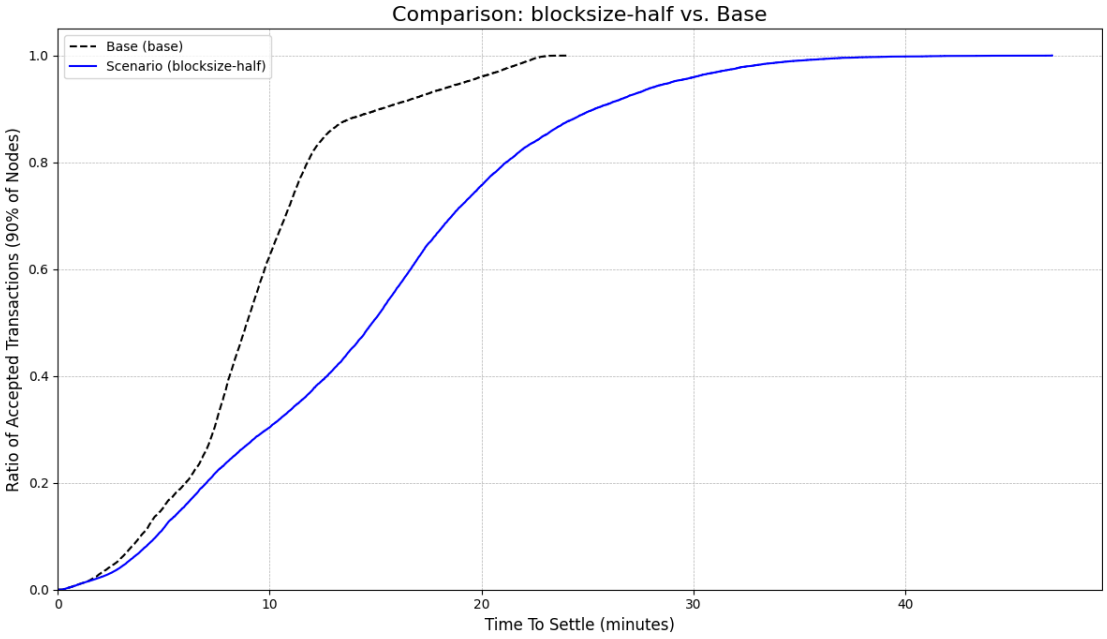

Figure 4.12: Comparison of finalization time with a halved block size (0.5 MB) vs. the baseline.

Changing Node Bandwidth (Network Throughput)

Node bandwidth determines how quickly blocks propagate across the network. As detailed in Table 4.12, insufficient bandwidth can harm not only speed but also consensus stability. This confirms a well-established principle, identified by Shahsavari et al. [63], Mivsic et al. [64], and Bistarelli et al. [61], that network delay is a primary cause of forks in Bitcoin.

Table 4.12: Performance Metrics for Network Throughput Variations.

Scenario	**Avg. Fork Rate (forks/run)**	**Median finalization Time (min)**	**90% finalization Time (min)**
Baseline	**0**	**8.25**	**13.42**
Increased Throughput	0	7.17	12.52
Decreased Throughput	2.0	10.99	21.00

Quantitative Impact on Finalization Increasing network throughput improved finalization times, reducing the 90% finalization time from **13.42 minutes** to **12.52 minutes**. Conversely, decreasing throughput hindered performance, increasing the 90% finalization time to **21.00 minutes**. More critically, the low-throughput scenario introduced significant consensus instability, resulting in an average of **2.0 forked blocks** per run. This outcome is strongly supported by research from Bistarelli et al. [61] and Nagayama et al. [65], who demonstrate that higher communication rates (lower delays) reduce fork probability, whereas Internet improvements such as increased bandwidth are key to reducing propagation delays and fork rates.

This instability arises because a decrease in bandwidth directly increases block propagation time. A threat to finality emerges when this propagation delay becomes comparable to the

average block discovery time (in our case, 9-10 minutes). If a node finds a new block before it has received the latest block from its peers, it will unknowingly build upon an older state of the chain. This creates a transient fork, and the network must expend time and resources to eventually agree on a single canonical chain. This confirms that sufficient bandwidth is critical not only for finalization speed but for maintaining a stable consensus and ensuring timely finality.

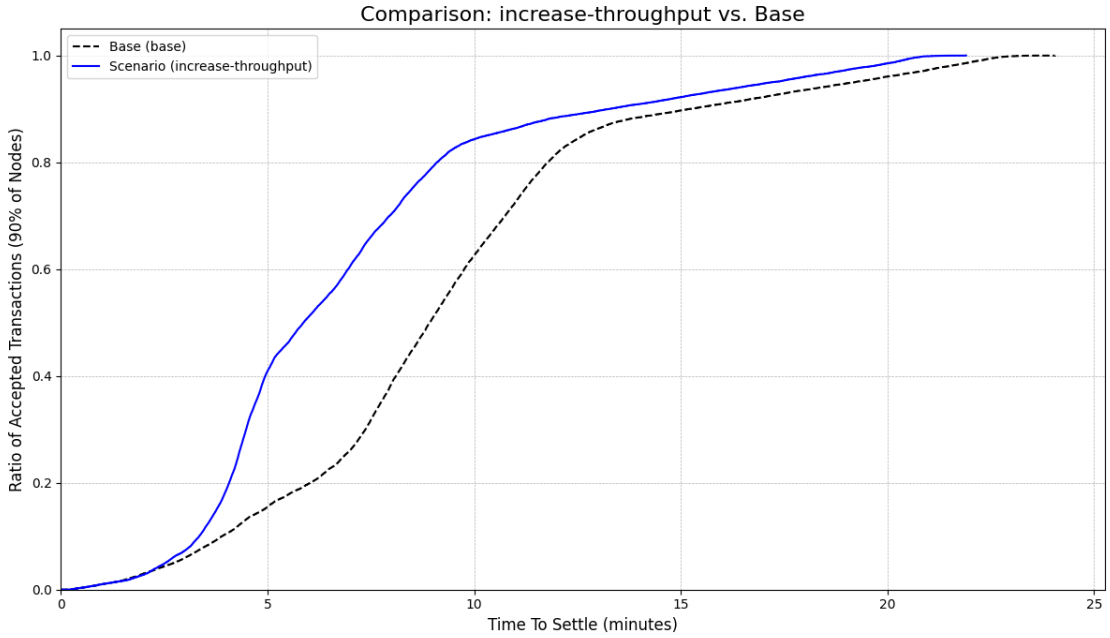

Figure 4.13: Comparison of finalization time with increased network throughput vs. the baseline.

The Impact of Transaction Generation Rate

The rate of new transaction arrivals determines the overall load on the network and the level of mempool congestion. Research by Mivsic et al. [66] supports this, showing that the mean number of transactions in a node's mempool increases super-linearly with the transaction

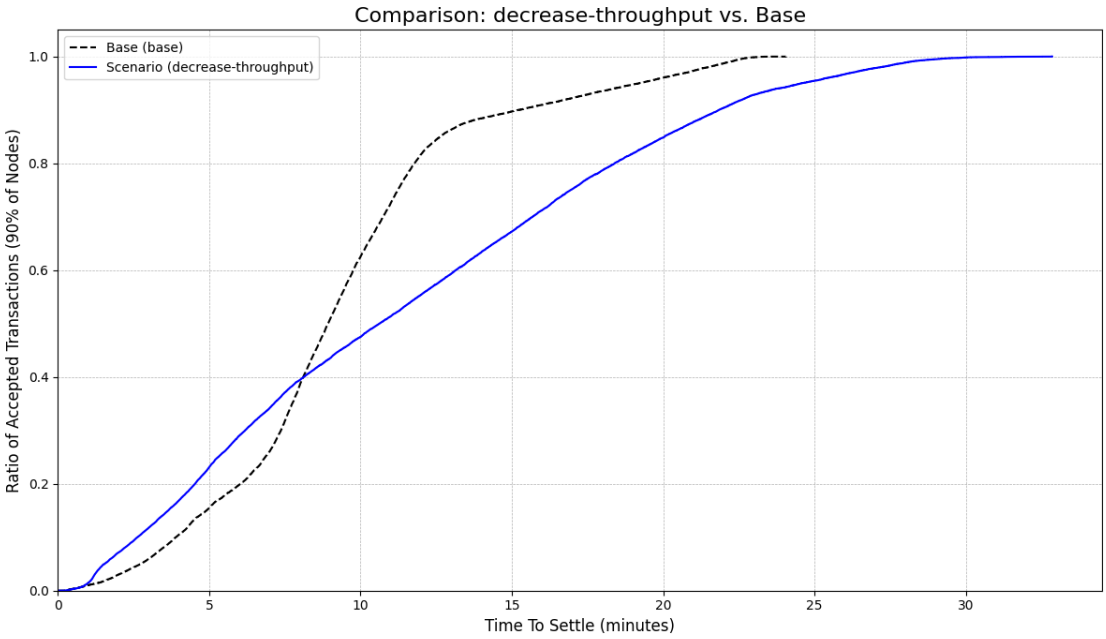

Figure 4.14: Comparison of finalization time with decreased network throughput vs. the baseline.

arrival rate. As shown in Table 4.13, a high arrival rate leads to a significant increase in the time required for transactions to be finalized.

Table 4.13: Performance Metrics for Transaction Rate Variations.

Scenario	**Median finalization Time (min)**	**90% finalization Time (min)**
Baseline	**8.25**	**13.42**
Increased Rate	25.88	38.78
Decreased Rate	7.86	11.49

Quantitative Impact on Finalization Increasing the transaction rate created congestion, causing the median finalization time to more than triple from **8.25 minutes** to **25.88 minutes**. The time to finalize 90% of transactions likewise increased from **13.42 minutes** to **38.78 minutes**. In contrast, decreasing the transaction rate alleviated congestion and

improved performance, reducing the 90% finalization time to **11.49 minutes**.

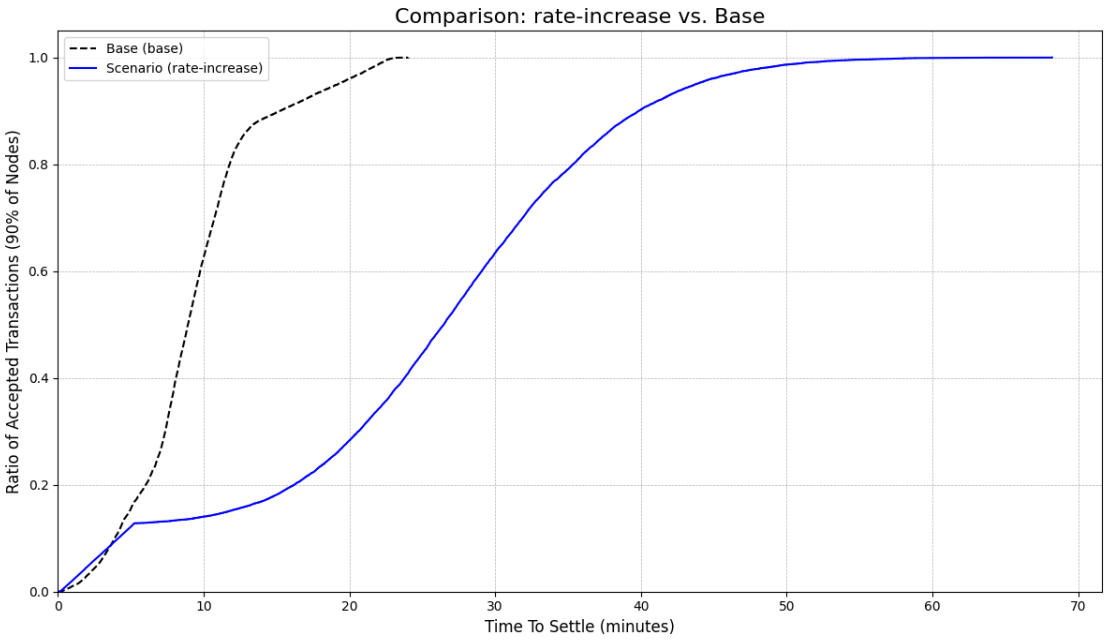

Figure 4.15: Comparison of finalization time with an increased transaction rate vs. the baseline.

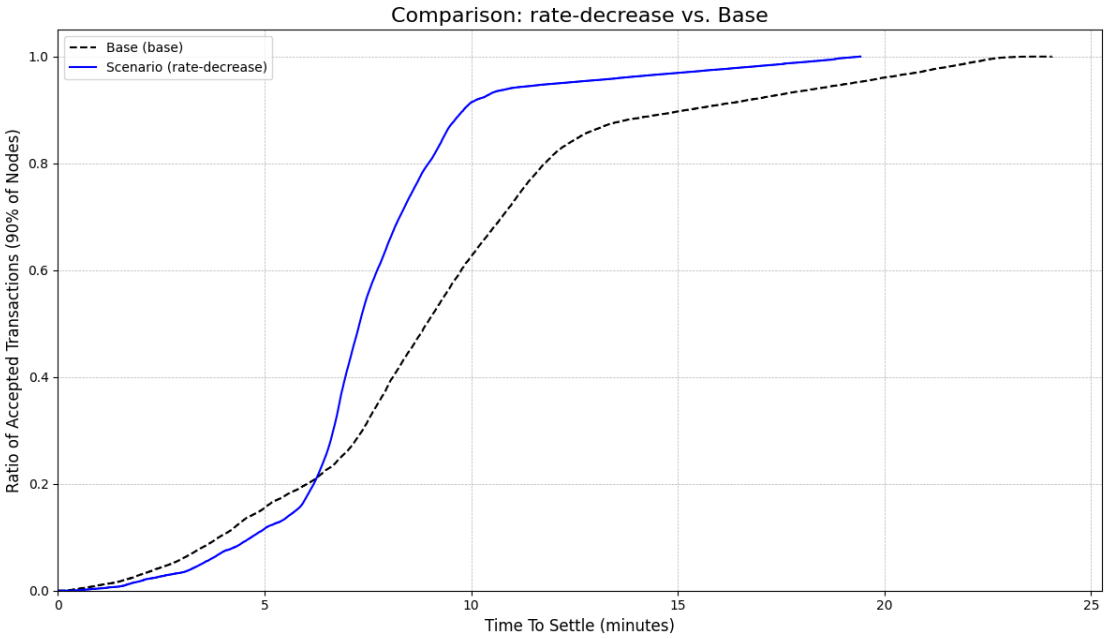

Figure 4.16: Comparison of finalization time with a decreased transaction rate vs. the baseline.

Overall Performance Comparison

For a comprehensive overview, Table 4.14 consolidates the finalization time data across all parameter variation scenarios. This table allows for a direct comparison of how each change impacted the time required to reach different finalization thresholds.

Table 4.14: Overall Summary: Time (in minutes) for transactions to reach finality across all scenarios.

Scenario	Time to Finalize 25%	Time to Finalize 50%	Time to Finalize 75%	Time to Finalize 90%	Time to Finalize 100%
base	**6.78**	**8.25**	**10.62**	**13.42**	**24.06**
increase-difficulty	8.79	15.44	20.20	24.52	40.08
decrease-difficulty	3.76	4.66	5.57	6.74	11.74
blocksize-double	6.37	7.97	9.73	12.84	22.53
blocksize-half	7.11	12.45	18.33	23.98	46.93
increase-throughput	4.53	7.17	9.90	12.52	21.90
decrease-throughput	4.81	10.99	16.85	21.00	32.85
rate-increase	18.87	25.88	32.50	38.78	68.22
rate-decrease	6.70	7.86	9.52	11.49	19.42

4.5 Chapter Summary

This chapter presented the experimental evaluation of the Bitcoin protocol using the CNSim framework. We first validated the simulator's fidelity by replicating baseline Bitcoin network conditions. We then systematically quantified the impact of malicious hash power on transaction finality and measured how system-wide performance changes in response to variations in key network and protocol parameters. The data and results presented here provide the empirical foundation for the broader discussion of our contributions and the implications of the finality-based methodology, which will be covered in the next chapter.

Chapter 5

Conclusions and Future Work

This thesis addressed the need for a more rigorous methodology for evaluating blockchain protocols, moving beyond ad-hoc metrics and informal heuristics. We introduced and applied a formal, simulation- and finality-based evaluation framework designed to assess blockchain consensus protocol performance against application-specific requirements. The framework consists of a formal definition of transaction finality, a way to measure it given simulated (or real) histories of consensus network operations, and an accompanying set of metrics, visualizations, and concrete analysis tasks. We operationalize this framework, we extended CNSim, an even-driven consensus network simulation, to allow generation of data that are usable for the proposed finality-based analysis. Using this improved tool, we conducted a systematic study of the Bitcoin protocol, validating its fidelity and using it to quantify the effects of malicious attacks and parameter variations on transaction finality; in the context of this analysis a novel extension of the simulator is used to simulate majority attacks. This

chapter consolidates the contributions and findings of this work, discusses the implications of our finality-based methodology, and outlines directions for future research.

5.1 Summary of Contributions

The contributions of this thesis are threefold: methodological, technical, and empirical.

1. **Methodological Contribution:** We proposed and validated a **finality-based evaluation framework** that re-frames performance analysis around user-defined service-level objectives (belief threshold d and time horizon t). This provides a formal, application-centric alternative to informal heuristics like the "six confirmation" rule, or fork rates, allowing for a more nuanced assessment of protocol suitability for specific use cases.

2. **Technical Contribution:** We enhanced the CNSim simulator with a **configurable implementation of complex adversarial strategies**, such as the Majority Attack. This was achieved by integrating the Strategy design pattern into the node architecture, which decoupled node behavior from the core logic. This enhancement provides a flexible framework for modeling and evaluating various state-dependent attack vectors, a capability the simulator previously lacked.

3. **Empirical Contribution:** We conducted a comprehensive experimental study of a Bitcoin-like network that (i) validated CNSim's fidelity against real-world network behavior, (ii) quantified the degradation of finality under varying levels of malicious hash power, and (iii) systematically measured the impact of key protocol parameters

on system-wide finalization performance.

5.2 Summary of Key Experimental Findings

Our experiments provided empirical validation for theoretical expectations and yielded new, quantitative insights into the practical performance of the Bitcoin protocol.

- **Attack resilience is a spectrum, not a binary state.** Our simulations of a targeted double-spend attack align with the theoretical analyses of Aponte-Novoa et al. [58] and Zhou et al. [60], but they also reveal a more nuanced reality. We found that attack impact scales directly with hashrate:
  - A low-power minority attack (**16.78%**) was insufficient to reverse the target transaction (100% finality success), but it still inflicted "collateral damage" by increasing the 90th percentile finalization time for all transactions from **12.74 to 16.36 minutes**.
  - A significant minority power (**29.68%**) introduced probabilistic failure, succeeding in its attack 20% of the time (80% finality success) and rendering finality unreliable, while pushing the 90th percentile finalization time to **19.49 minutes**.
  - As expected, a majority attacker (**66.25%**) was deterministically successful (0% finality success) and had a catastrophic impact on system-wide performance, more than quadrupling the 90th percentile finalization time to **59.63 minutes**.
- **Protocol parameters present critical and predictable trade-offs.** The impact

of parameter changes on performance was consistent with established probability and queueing models of blockchain systems, such as those developed by Mivsic et al. [66] and Shahsavari et al. [63].

- **Mining Difficulty:** Lowering difficulty dramatically improved performance, halving the median finalization time from 8.25 to **4.66 minutes**. However, this comes at the theoretical risk of increasing forks and compromising consensus stability, a finding supported by the work of Bistarelli et al. [61].
- **Network Capacity:** The interplay between block size and transaction rate functioned as a clear bottleneck. Halving the block size to 0.5 MB severely hindered performance, increasing the 90th percentile finalization time to **23.98 minutes**. Conversely, doubling the block size only yielded a minor improvement because the system was not congested at the baseline transaction rate.
- **Network Throughput:** Insufficient network bandwidth proved to be a direct cause of consensus instability. Decreasing throughput not only slowed finalization (90th percentile time increased to **21.00 minutes**) but also introduced an average of **2.0 forked blocks** per run, confirming that timely finality depends fundamentally on both computation and communication [65].

5.3 Discussion: The Power of a Finality-Based Methodology

The primary directive from the analysis in Chapter 4 was to expand upon the utility of our evaluation methodology. The framework's intended value lies in its ability to translate vast, complex simulation data into clear, interpretable, and comparable results, allowing for a deeper understanding of protocol dynamics. In this section, addressing RQ4, we reflect on our experience from using the framework in our Bitcoin study in order to draw some initial assessment of its feasibility and value.

Revealing Clear Trade-offs and Bottlenecks. The framework excels at making performance trade-offs explicit. For example, by observing the Aggregate finalization Curve under different mining difficulties (Figures 4.9 and 4.10), we could precisely quantify the trade-off between finalization speed and block production rate. Furthermore, the framework was instrumental in diagnosing the root cause of performance degradation. Our results showed that halving the block size or decreasing network throughput created severe bottlenecks whose negative impact was far more pronounced than the modest gains from increasing them. For instance, doubling the block size only reduced the 90% finalization time by **4.3%** (from 13.42 to 12.84 minutes) because the system was not constrained by block space at the baseline workload. This highlights the framework's ability to identify the true limiting factors in a system.

Connecting Performance to Consensus Instability. A key insight enabled by our methodology was the direct, empirical link between performance bottlenecks and consensus stability. In the low-throughput scenario, the Aggregate finalization Curve (Figure 4.14) did not just show slower confirmations; when correlated with chain stability metrics, it revealed that this slowdown was accompanied by an average of **2.0 forked blocks** per run. The framework thus allowed us to demonstrate causality: a communication bottleneck directly undermined the stability of the consensus mechanism itself. This holistic view, connecting application-level performance (finality) to protocol-level health (forks), is essential for robust protocol design and analysis.

The Complementary Power of System-wide and Single-Transaction Metrics The finality framework provides a multi-faceted view of performance by combining different lenses. The **Confidence Plot** for a single transaction (e.g., Figure 4.7) proved to be an invaluable diagnostic tool, showing not just *if* an attack succeeded, but precisely *how* it unfolded by visualizing the dramatic collapse of consensus. However, observing a single transaction in isolation can be misleading. It was the **Aggregate finalization Curve** (e.g., Figure 4.4) that revealed the systemic "collateral damage" of a failed minority attack on the entire network. The former is a powerful tool for micro-level diagnostics, while the latter provides a stable, holistic "fingerprint" of macro-level network health. The ability to use both in concert is a core strength of this methodology.

The Role of Simulation in Understanding Realistic System Behavior. Finally, it is important to reflect on the role of the simulator itself. While analytical models can provide theoretical guarantees, they often rely on simplifying assumptions, abstracting away the complexities of network latency, node heterogeneity, and sophisticated, state-dependent adversarial strategies. A simulation framework like CNSim is essential precisely because it allows us to relax these assumptions and observe the protocol's emergent behavior under more realistic conditions. The process of building and running these simulations provided a far more intimate understanding of how factors like block propagation delays and mempool congestion create the conditions for forks or degrade finality. Phenomena such as the systemic "collateral damage" of a failed attack are difficult to capture in a purely analytical model but become readily apparent in simulation, underscoring the necessity of simulation for a comprehensive and practical evaluation of consensus protocols.

5.4 Limitations and Future Work

This study, while comprehensive in its approach, has limitations that naturally lead to avenues for future research. Our models for adversarial behavior and network topology were simplifications.

Based on this, future work should proceed in several directions:

- **Comparative Protocol Analysis:** The immediate next step is to apply the finality framework to other consensus mechanisms (e.g., Proof-of-Stake protocols like Casper or Tendermint) and ledger structures (e.g., DAGs). Implementing these in CNSim would

enable the first true "apples-to-apples" comparison of their finality characteristics under identical conditions.

- **Richer Adversary and Economic Models:** Future work should incorporate more sophisticated adversary models, including economically rational agents who might engage in selfish mining, a strategy introduced by Eyal and Sirer [13]. Modeling transaction fee market dynamics would also yield deeper insights into transaction inclusion policies and potential fee-based vulnerabilities.

- **Analysis of Layer-2 Solutions:** An important research direction is to extend the simulation to model the interaction between the base blockchain (Layer 1) and scaling solutions like the Lightning Network, analyzing their collective impact on end-user transaction finality and security.

- **Validation Against Analytical Models:** While this thesis focused on simulation-based evaluation, it is important to acknowledge the existence of foundational analytical models for finality. In the original Bitcoin paper, Satoshi Nakamoto modeled the race between an attacker and the honest chain as a Gambler's Ruin problem. The probability of an attacker successfully catching up from a deficit of z blocks is calculated based on their share of the total hash power, assuming block generation follows a Poisson process. A thorough, quantitative validation of our simulation results against Nakamoto's analytical model was beyond the scope of this work. However, such a comparison represents a compelling avenue for future research. It would not only

help validate the simulator's fidelity but also test the robustness of the analytical model's assumptions against the more complex and less ideal conditions introduced in a simulation, such as network latency and variable block propagation times.

5.5 Concluding Remarks

In conclusion, this thesis has argued for and demonstrated a systematic, finality-based approach to blockchain evaluation. By supporting this methodology with flexible and extensible simulation tools, we can move beyond heuristics toward a more rigorous, empirical, and comparative science of distributed consensus. The insights gained from this work can guide developers and network operators in tuning parameters and designing protocols to build more robust, secure, and performant decentralized systems.

Bibliography

[1] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Satoshi Nakamoto* (2008).

[2] Muhammad Saad et al. "Revisiting nakamoto consensus in asynchronous networks: A comprehensive analysis of bitcoin safety and chainquality". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 988–1005.

[3] Ren Zhang and Bart Preneel. "Lay down the common metrics: Evaluating proof-of-work consensus protocols' security". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 175–192.

[4] Vitalik Buterin et al. "Ethereum white paper". In: *GitHub repository* 1.22-23 (2013), pp. 5–7.

[5] Serguei Popov. "The tangle". In: *White paper* 1.3 (2018), p. 30.

[6] Carol Coye Benson and Scott Loftesness. *Payments systems in the US: A guide for the payments professional*. Glenbrook Partners, 2013.

[7] Taishi Nakai et al. "The blockchain trilemma described by a formula". In: *2023 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2023, pp. 41–46.

[8] Joseph Bonneau et al. "Why buy when you can rent ? Bribery attacks on Bitcoin consensus". In: 2015. URL: https://api.semanticscholar.org/CorpusID:14164538.

[9] Kevin Liao and Jonathan Katz. "Incentivizing Double-Spend Collusion in Bitcoin". In: 2017. URL: https://api.semanticscholar.org/CorpusID:10174917.

[10] Vojislav B. Mišić, Jelena V. Misic, and Xiaolin Chang. "On Forks and Fork Characteristics in a Bitcoin-Like Distribution Network". In: *2019 IEEE International Conference on Blockchain (Blockchain)* (2019), pp. 212–219. URL: https://api.semanticscholar.org/CorpusID:209853576.

[11] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. "A Theoretical Model for Fork Analysis in the Bitcoin Network". In: *2019 IEEE International Conference on Blockchain (Blockchain)* (2019), pp. 237–244. URL: https://api.semanticscholar.org/CorpusID:209853692.

[12] Luca Ambrosini, Matija Pikorec, and Claudio Juan Tessone. "Visualization of Blockchain Consensus Degradation". In: *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (2022), pp. 1–2. URL: https://api.semanticscholar.org/CorpusID:250118743.

[13] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *Communications of the ACM* 61.7 (2018), pp. 95–102.

[14] Meni Rosenfeld. "Analysis of hashrate-based double spending". In: *arXiv preprint arXiv:1402.2009* (2014).

[15] Rui Zhang, Rui Xue, and Ling Liu. "Security and privacy on blockchain". In: *ACM Computing Surveys (CSUR)* 52.3 (2019), pp. 1–34.

[16] Marianna Belotti et al. "A vademecum on blockchain technologies: When, which, and how". In: *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3796–3838.

[17] Andreas M Antonopoulos and David A Harding. *Mastering bitcoin.* " O'Reilly Media, Inc.", 2023.

[18] Arvind Narayanan et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, 2016.

[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol: Analysis and applications". In: *Journal of the ACM* 71.4 (2024), pp. 1–49.

[20] Rafael Pass, Lior Seeman, and Abhi Shelat. "Analysis of the blockchain protocol in asynchronous networks". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2017, pp. 643–673.

[21] Vitalik Buterin and Virgil Griffith. "Casper the friendly finality gadget". In: *arXiv preprint arXiv:1710.09437* (2017).

[22] Jing Chen and Silvio Micali. "Algorand". In: *arXiv preprint arXiv:1607.01341* (2016).

[23] Jae Kwon. "Tendermint: Consensus without mining". In: *Draft v. 0.6, fall* 1.11 (2014), pp. 1–11.

[24] Ethan Buchman. "Tendermint: Byzantine fault tolerance in the age of blockchains". PhD thesis. University of Guelph, 2016.

[25] Christian Decker and Roger Wattenhofer. "Information propagation in the bitcoin network". In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

[26] Rainer Böhme et al. "Bitcoin: Economics, technology, and governance". In: *Journal of economic Perspectives* 29.2 (2015), pp. 213–238.

[27] Sarah Meiklejohn et al. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 127–140.

[28] Joshua A Kroll, Ian C Davey, and Edward W Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries". In: *Proceedings of WEIS*. Vol. 2013. 11. Citeseer. 2013.

[29] Joseph Bonneau et al. “Sok: Research perspectives and challenges for bitcoin and cryptocurrencies”. In: *2015 IEEE symposium on security and privacy*. IEEE. 2015, pp. 104–121.

[30] John R Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[31] Ethan Heilman et al. “Eclipse attacks on {Bitcoin’s}{peer-to-peer} network”. In: *24th USENIX security symposium (USENIX security 15)*. 2015, pp. 129–144.

[32] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. “Hijacking bitcoin: Routing attacks on cryptocurrencies”. In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 2017, pp. 375–392.

[33] Arthur Gervais et al. “On the security and performance of proof of work blockchains”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 3–16.

[34] Shehar Bano et al. “SoK: Consensus in the age of blockchains”. In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, pp. 183–198.

[35] Kyle Croman et al. “On Scaling Decentralized Blockchains: (A Position Paper)”. In: *International conference on financial cryptography and data security*. Springer. 2016, pp. 106–125.

[36] Adem Efe Gencer et al. “Decentralization in bitcoin and ethereum networks”. In: *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer. 2018, pp. 439–457.

[37] Carlos Faria and Miguel Correia. “BlockSim: blockchain simulator”. In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2019, pp. 439–446.

[38] Seyed Mehdi Fattahi, Adetokunbo Makanju, and Amin Milani Fard. “SIMBA: An efficient simulator for blockchain applications”. In: *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE. 2020, pp. 51–52.

[39] Deepak Kumar Gouda, Shashwat Jolly, and Kalpesh Kapoor. “Design and validation of blockeval, a blockchain simulator”. In: *2021 international conference on communication systems & networks (COMSNETS)*. IEEE. 2021, pp. 281–289.

[40] Yusuke Aoki et al. “Simblock: A blockchain network simulator”. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 325–329.

[41] Lyubomir Stoykov, Kaiwen Zhang, and Hans-Arno Jacobsen. “Vibes: fast blockchain simulations for large-scale peer-to-peer networks”. In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*. 2017, pp. 19–20.

[42] Aditya Deshpande, Pezhman Nasirifard, and Hans-Arno Jacobsen. “eVIBES: configurable and interactive ethereum blockchain simulation framework”. In: *Proceedings of the 19th International Middleware Conference (Posters)*. 2018, pp. 11–12.

[43] Habib Yajam, Elnaz Ebadi, and Mohammad Ali Akhaee. “JABS: A Blockchain Simulator for Researching Consensus Algorithms”. In: *IEEE Transactions on Network Science and Engineering* (2023).

[44] Mohammadreza Rasolroveicy, Wejdene Haouari, and Marios Fokaefs. “BlockCompass: A benchmarking platform for blockchain performance”. In: *IEEE Transactions on Computers* (2024).

[45] Tien Tuan Anh Dinh et al. “Blockbench: A framework for analyzing private blockchains”. In: *Proceedings of the 2017 ACM international conference on management of data*. 2017, pp. 1085–1100.

[46] Wonseok Choi and James Won-Ki Hong. “Performance evaluation of ethereum private and testnet networks using hyperledger caliper”. In: *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2021, pp. 325–329.

[47] Dylan Yaga et al. “Blockchain technology overview”. In: *arXiv preprint arXiv:1906.11078* (2019).

[48] Nathan Sealey, Adnan Aijaz, and Ben Holden. “Iota tangle 2.0: Toward a scalable, decentralized, smart, and autonomous IoT ecosystem”. In: *2022 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE. 2022, pp. 01–08.

[49] Miguel Castro, Barbara Liskov, et al. “Practical byzantine fault tolerance”. In: *OsDI*. Vol. 99. 1999. 1999, pp. 173–186.

[50] S. Liaskos, A. Radjou, and D.S. Muhammad. *Finality-oriented simulation-based evaluation and comparison of blockchain consensus networks: the CNSim framework*. Tech. rep. York University, 2025. URL: `https://hdl.handle.net/10315/42938`.

[51] Widya Nita Suliyanti and Riri Fitri Sari. “Evaluation of hash rate-based double-spending based on proof-of-work blockchain”. In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE. 2019, pp. 169–174.

[52] Erich Gamma et al. “Design patterns: Abstraction and reuse of object-oriented design”. In: *ECOOP’93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*. Springer. 1993, pp. 406–431.

[53] Blockchain.com. *Bitcoin Mining Pool Distribution*. Data reflects hashrate distribution over the last 10 days. 2025. URL: `https://www.blockchain.com/explorer/charts/pools` (visited on 02/11/2025).

[54] CoinWarz. *Bitcoin Difficulty Chart*. 2025. URL: `https://www.coinwarz.com/mining/bitcoin/difficulty-chart` (visited on 02/11/2025).

[55] Blockchain.com. *Bitcoin Network Charts & Data*. Source for transaction arrival rate, transaction fees, and BTC-USD exchange rate. 2025. URL: `https://www.blockchain.com/explorer/charts` (visited on 02/10/2025).

[56] Bitcoin Visuals. *Chain Tx Size*. `https://bitcoinvisuals.com/chain-tx-size`.

[57] Blockchair. *Bitcoin Blocks Data Dumps*. Used for real-world block interval data from February 2024. 2025. URL: `https://gz.blockchair.com/bitcoin/blocks/` (visited on 02/11/2025).

[58] Fredy Andres Aponte-Novoa et al. "The 51% attack on blockchains: A mining behavior study". In: *IEEE access* 9 (2021), pp. 140549–140564.

[59] Haoran Zhu et al. "Delay impact on stubborn mining attack severity in imperfect Bitcoin network". In: *IEEE Transactions on Network Science and Engineering* 11.3 (2023), pp. 2586–2595.

[60] Chencheng Zhou et al. "System-Level Dependability Analysis of Bitcoin under Eclipse and 51% Attacks". In: *International Journal of Mathematical, Engineering and Management Sciences* 8.4 (2023), p. 547.

[61] Stefano Bistarelli et al. "Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays". In: *Concurrency and Computation: Practice and Experience* 35.16 (2023), e6749.

[62] Tong Cao et al. "Characterizing the impact of network delay on bitcoin mining". In: *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2021, pp. 109–119.

[63] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. "A theoretical model for block propagation analysis in bitcoin network". In: *IEEE Transactions on Engineering Management* 69.4 (2020), pp. 1459–1476.

[64] Jelena Mišić et al. "Modeling of bitcoin's blockchain delivery network". In: *IEEE Transactions on Network Science and Engineering* 7.3 (2019), pp. 1368–1381.

[65] Ryunosuke Nagayama, Ryohei Banno, and Kazuyuki Shudo. "Identifying impacts of protocol and internet development on the bitcoin network". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–6.

[66] Jelena Mišić, Vojislav B Mišić, and Xiaolin Chang. "Performance of bitcoin network with synchronizing nodes and a mix of regular and compact blocks". In: *IEEE Transactions on Network Science and Engineering* 7.4 (2020), pp. 3135–3147.