Trustworthy Foundations for Web3

Nasrulin, B.

DOI
10.4233/uuid:f6424b27-b9a4-47e2-9219-7088b547d5e3

Publication date
2025

Document Version
Final published version

Citation (APA)
Nasrulin, B. (2025). *Trustworthy Foundations for Web3*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:f6424b27-b9a4-47e2-9219-7088b547d5e3

B. A. NASRULIN

Trustworthy Foundations for Web3

Trustworthy Foundations for Web3

Trustworthy Foundations for Web3

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
friday 26 september 2025 at 10:00 o'clock

by

Bulat Aydarovich NASRULIN

Master of Informatics and Computer Engineering,
Innopolis University, Russia,
born in Tashkent, Uzbekistan.

This dissertation has been approved by the promotors.

promotor: Prof. dr. ir. D.H.J. Epema
promotor: Dr. ir. J.A. Pouwelse

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. D.H.J. Epema,	Delft University of Technology, *promotor*
Dr. ir. J.A. Pouwelse,	Delft University of Technology, *promotor*
Independent members:	
Prof. dr. M.M. de Weerdt,	Delft University of Technology
Prof. dr. ir. N. Bharosa,	Delft University of Technology
Prof. dr. D.G.J. Bongaerts,	RSM, Erasmus University
Dr. B. Özkan,	Delft University of Technology
Dr. D. Frey,	IRISA/Inria, France

This work was carried out in the ASCI graduate school. ASCI dissertation series number 477.

Keywords:	blockchain, decentralization, gossip, benchmarking, peer-to-peer, reputation, Sybil, Web3
Printed by:	ridderprint.nl
Cover:	Amal Nasrulina and ChatGPT. The cover illustrates the importance of cooperation as an homage to the Bruegel's Tower of Babel.
Style:	TU Delft House Style, with modifications by Moritz Beller `https://github.com/Inventitech/phd-thesis-template`

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.

ISBN 978-94-6518-108-0

An electronic version of this dissertation is available at `http://repository.tudelft.nl/`.

I've since learned that it's a hell of a lot easier to just build something than to try to convince somebody who doesn't believe it's possible

Paul Baran, Internet Pioneer

Contents

Summary

Web3 represents an ambitious attempt to democratize access to financial instruments and innovate economic activity. At its core, the Web3 aims to build a public infrastructure where users actively contribute and share control. Realizing this vision requires overcoming a number of difficult challenges, such as infrastructure reliability, participant accountability, resilience to attacks, and incentive compatibility. This thesis addresses these issues by investigating the performance, scalability, accountability, and cooperative behaviors in blockchain systems. Our goal is to develop and evaluate mechanisms that meet these challenges, thereby providing a blueprint for the future of decentralized systems.

In Chapter 1, we outline the core challenges and considerations for creating a trustworthy Web3 foundation, focusing on reliability, accountability, Sybil attack resilience, and incentive compatibility. We discuss the evolution of decentralized systems and formulate the primary research question of this thesis along with four derivative research questions. Additionally, we describe our research methodology and outline the structure and contributions of this thesis.

In Chapter 2, we present the Gromit benchmarking framework, designed to analyze the performance and scalability of blockchain systems. Our study highlights the performance limitations of state-of-the-art blockchain implementations, particularly under real-world conditions. We conclude that realistic benchmarking is essential for identifying performance bottlenecks and guiding the optimization of blockchain systems.

In Chapter 3, we introduce the LØ pre-consensus protocol to enhance accountability among blockchain validators. By logging all received transactions into a secure mempool data structure, LØ enhances transparency and detects manipulative behaviors such as censorship, reordering, or injection. We demonstrate that adding accountability mechanisms before the consensus phase significantly reduces manipulative behaviors and enhances trust in blockchain systems.

In Chapter 4, we develop MeritRank, a novel Sybil-tolerant feedback aggregation mechanism for reputation systems. Unlike traditional approaches that attempt to prevent Sybil attacks outright, MeritRank limits the benefits that attackers can derive from such attacks. Our simulations show that MeritRank effectively reduces the impact of Sybil attacks, ensuring that attackers gain benefits only proportional to their genuine contributions.

In Chapter 5, we design an incentive-compatible transaction dissemination protocol, Coop, for permissionless blockchains. Our protocol aligns individual incentives with collective goals, encouraging cooperative behavior while detecting and isolating freeriders and selfish peers. We conclude that Coop is a simple yet effective incentive mechanism that fosters cooperation and minimizes selfish behavior in decentralized systems.

In Chapter 6, we summarize our findings and provide overarching conclusions of this thesis. We highlight the importance of combining accountability mechanisms and performance benchmarking to build reliable blockchain systems, and the synergy between

reputation and incentive systems for fostering a resilient Web3 ecosystem. We also outline several promising directions for futdure research, including enhancing benchmarking frameworks, extending accountability mechanisms, improving Sybil-tolerant reputation systems, and exploring broader applications of incentive-compatible mechanisms. These future directions aim to further advance the development of a trustworthy and resilient Web3 ecosystem.

Samenvatting

Naarmate digitale technologie zich ontwikkelt, biedt een gedecentraliseerd, door gebruikers gestuurd internet—vaak Web3 genoemd—een van de meest ambitieuze mogelijkheden voor de toekomst. In tegenstelling tot traditionele infrastructuren, die gedomineerd worden door grote bedrijven en overheden, voorziet Web3 een digitale omgeving waarin gebruikers actief bijdragen aan en controle hebben over publieke infrastructuur. Om deze visie te realiseren, moeten verschillende uitdagingen worden overwonnen, zoals betrouwbaarheid, verantwoordelijkheid, weerbaarheid tegen Sybil-aanvallen en prikkelcompatibiliteit. Dit proefschrift onderzoekt deze fundamentele kwesties en biedt waardevolle inzichten voor de opbouw van een veerkrachtig en betrouwbaar Web3-ecosysteem.

Web3-oplossingen zijn nog volop in ontwikkeling en er blijven veel uitdagingen onontgonnen. Dit proefschrift behandelt deze onderwerpen door de prestaties, schaalbaarheid, verantwoordelijkheid en coöperatieve gedragingen van blockchainsystemen te onderzoeken. Het doel is om mechanismen te ontwikkelen en te evalueren die deze uitdagingen aanpakken, en zo een blauwdruk te bieden voor de toekomst van gedecentraliseerde systemen.

In dit proefschrift zetten we belangrijke stappen in het definiëren en aanpakken van de kritieke uitdagingen voor Web3. We beschrijven en evalueren vier prototype-mechanismen, die elk zijn ontworpen om specifieke aspecten van de basisproblemen van Web3 aan te pakken. Onze aanpak is verkennend, met als doel toekomstig onderzoek en ontwikkeling te sturen in de richting van een betrouwbaar en veerkrachtig Web3-ecosysteem.

In Hoofdstuk 1 schetsen we de kernuitdagingen en aandachtspunten voor het creëren van een betrouwbaar Web3-fundament, met de nadruk op betrouwbaarheid, verantwoordelijkheid, veerkracht tegen Sybil-aanvallen en prikkelcompatibiliteit. We bespreken de ontwikkeling van gedecentraliseerde systemen en formuleren de hoofdonderzoeksvraag van dit proefschrift, samen met vier afgeleide onderzoeksvragen. Daarnaast beschrijven we onze onderzoeksmethodologie en geven we een overzicht van de structuur en bijdragen van dit proefschrift.

In Hoofdstuk 2 introduceren we het Gromit-benchmarkingframework, ontworpen om de prestaties en schaalbaarheid van blockchainsystemen te analyseren. Ons onderzoek toont de prestatiebeperkingen van geavanceerde blockchain-implementaties aan, vooral onder realistische omstandigheden. We concluderen dat realistische benchmarking essentieel is om prestatieknelpunten te identificeren en de optimalisatie van blockchainsystemen te begeleiden.

In Hoofdstuk 3 introduceren we het LØ-preconsensusprotocol om de verantwoordelijkheid onder blockchain-validators te versterken. Door alle ontvangen transacties in een beveiligde mempool-datastructuur vast te leggen, vergroot LØ de transparantie en detecteert het manipulatief gedrag zoals censuur, herschikking of injectie. We tonen aan dat het toevoegen van verantwoordingsmechanismen vóór de consensusfase manipulatief gedrag aanzienlijk vermindert en het vertrouwen in blockchainsystemen versterkt.

In Hoofdstuk 4 ontwikkelen we MERITRANK, een nieuw Sybil-tolerant feedback - aggregatiemechanisme voor reputatiesystemen. In tegenstelling tot traditionele benaderingen die Sybil-aanvallen proberen te voorkomen, beperkt MERITRANK de voordelen die aanvallers uit dergelijke aanvallen kunnen halen. Onze simulaties laten zien dat MERITRANK de impact van Sybil-aanvallen effectief vermindert, zodat aanvallers alleen voordelen behalen die in verhouding staan tot hun werkelijke bijdragen.

In Hoofdstuk 5 ontwerpen we een prikkelcompatibel transactiedistributieprotocol, COOP, voor permissieloze blockchains. Ons protocol stemt individuele prikkels af op collectieve doelen, waardoor coöperatief gedrag wordt aangemoedigd terwijl freeriders en egoïstische peers worden gedetecteerd en geïsoleerd. We concluderen dat COOP een eenvoudig maar effectief prikkelmechanisme is dat samenwerking bevordert en egoïstisch gedrag in gedecentraliseerde systemen minimaliseert.

In Hoofdstuk 6 vatten we onze bevindingen samen en presenteren we de overkoepelende conclusies van dit proefschrift. We benadrukken het belang van het combineren van verantwoordingsmechanismen en prestatiebenchmarking om betrouwbare blockchainsystemen te bouwen, evenals de synergie tussen reputatie- en prikkelsystemen voor het bevorderen van een veerkrachtig Web3-ecosysteem. Tot slot schetsen we verschillende veelbelovende richtingen voor toekomstig onderzoek, waaronder het verbeteren van benchmarkingframeworks, het uitbreiden van verantwoordingsmechanismen, het verbeteren van Sybil-tolerante reputatiesystemen en het verkennen van bredere toepassingen van prikkelcompatibele mechanismen. Deze toekomstige richtingen zijn bedoeld om de ontwikkeling van een betrouwbaar en veerkrachtig Web3-ecosysteem verder te bevorderen.

Acknowledgments

I have chosen the quote in the header as something that reflects my journey and experience. In academia we often fall in love with theoretical elegance, convincing ourselves that we understand exactly how things work. In practice, however, the world is different. Much of my PhD was spent persuading respected researchers that our models are incomplete, that they have flaws, and that alternative approaches can succeed. It is heartening to discover that I was not alone in this, and that questioning assumptions is an essential part of every new technological emergence.

I was fortunate to have unwavering support throughout this journey. I owe my deepest gratitude to my two promotors, whose complementary strengths guided me every step of the way: Johan Pouwelse and Dick Epema. Johan taught me the power of intuition and perseverance. He showed me how to appreciate the combination of scientific rigor and hard work, and to pursue ambitious, practical problems even when many scientists give up at the first sign of difficulty. Dick shared with me the rich history and recurring motifs of computer science. Every discussion with him sharpened my ability to read papers meticulously and inspired me to strive for clarity and depth in my own writing.

I also thank my colleagues and co-authors in the lab, who were invaluable brainstorming partners. Georgy introduced me to inspiring industry conferences, connected me with key people, and helped distill the essence of my research ideas. Jeremie supported me endlessly in writing papers and shared both professional insights and life advice. Can listened patiently to my rambles and engaged in countless stimulating discussions. To Martijn, whose work ethic and positive attitude were a joy to witness. To Quinten, whose sense of humor brightened every meeting. To Rowdy, for your enthusiastic encouragement of my ideas and friendship. To Vadim, for helping me out and for great brainstorm sessions. And to Marcel, Petru, Aditya, Fuad, Jan, Sandip, Alexander and Andrey – thank you for the many rounds of coffee and the great conversations. I would like to thank Kim and Sophie for their help with administrative matters.

A special thanks to my friends Pavel, Kirill and Rauf, who persistently asked when I would finally finish this dissertation.

Finally, to my beloved wife for her infinite patience and support, and to my parents and sister for encouraging me to pursue this path.

Bulat

1

Introduction

At its core, Web3 relies on blockchain technology, a distributed ledger that ensures data integrity and transparency through cryptographic hashing and consensus algorithms [1]. Decentralized applications are constructed using smart contracts, which are invoked and controlled by users. These users are required to sign all their transactions with cryptographic keys. These elements aim to achieve desirable properties such as decentralization, immutability, transparency, security, and user sovereignty. Decentralization mitigates single points of failure and potential censorship; immutability ensures that once data is recorded, it cannot be altered; transparency allows for open verification of all transactions; security safeguards data and transactions through cryptographic means; and user sovereignty emphasizes control over one's data and identity. Together, these properties form the basis for the *Foundation for Web3*.

This thesis explores the theoretical promise and practical challenges of building a trustworthy foundation for Web3. While the decentralized architecture of Web3 theoretically ensures desirable properties such as transparency, security, and user sovereignty, it meets with significant challenges in practice. Vulnerabilities in peer-to-peer networks, risks of transaction manipulations by the participants, and general unreliability of the systems highlight the complexity of achieving true trustworthiness. These challenges underscore the need for a rigorous examination of the Web3 foundation, which this thesis aims to provide.

Within this thesis, we discuss the current shortcomings of Web3, with a special focus on its foundations. We identify four prominent areas for improvement: reliability, accountability, Sybil attack resilience, and incentive compatibility. We design and analyze novel mechanisms that directly address these issues. Specifically, we conduct a benchmarking study, design an accountable message dissemination mechanism, develop a Sybil-tolerant reputation algorithm, and create a decentralized scoring system to mitigate the effects of selfish nodes. We now explore the history and design philosophies of Web3 (Section 1.1), define what constitutes the foundation of Web3 (Section 1.2), current issues in Web3 (Section 1.3), present our research questions (Section 1.4), outline our research methodology (Section 1.5), and describe the structure of this thesis along with its scientific contributions (Section 1.6).

1

1.1 What is Web3?

Web3 is presented as the next evolutionary phase of the internet [2], frequently referred to as the "Decentralized Web" [3]. Initially conceived as a technological underpinning for cryptocurrencies [1], the scope of Web3 has expanded to include a wide array of so called 'decentralized applications' or DApps. More than a mere technological change, Web3 proposes a conceptual transformation in the foundational architecture of the internet. It aims to foster a more transparent and open digital infrastructure, drastically reducing reliance on centralized intermediaries. The vision of Web3 is to pave the way for a decentralized infrastructure that facilitates direct interactions and transactions between the users with enhanced security and privacy.

To better understand Web3, this section first contextualizes it within the scope of decentralized systems and the broader evolution of Web technologies. This includes tracing its historical path, understanding the preconditions for its inception, and section its potential. Furthermore, this introduction outlines the core design philosophies intrinsic to Web3.

1.1.1 Evolution of Decentralized Systems

The concept of decentralization is not new and has its roots in early computer science and network design. Historically, research in computer science has oscillated between centralized and decentralized paradigms.

In the early days of computer science, the field was dominated by mainframes, serving as central processors and repositories of data [4]. Yet their centralization posed notable risks, such as system-wide disruptions from malfunctions. Besides centralization, mainframes were often task-specific and inflexible, making system adaptations and integrations time-consuming and complex. Their proprietary nature tied organizations to specific vendors, limiting competition and increasing costs [5]. Despite their power, mainframes required substantial investments for scalability, often necessitating entirely new systems or extensive upgrades. This led to high operational costs due to the expense of maintaining, upgrading, and supporting the hardware and software infrastructure. These systems also demanded specialized personnel for operation and maintenance, further adding to the overall cost. Mainframes relied heavily on batch processing, resulting in delayed user feedback, and their basic user interfaces restricted interaction. Moreover, while their central nature provided some security benefits, it also meant that a single breach could compromise the entire system, especially as user connectivity expanded.

In 1964, Paul Baran described decentralization in his seminal work "On Distributed Communications Networks" [6]. Baran defined three types of network architectures: centralized, decentralized, and distributed. In a centralized network, a single central node manages the communication between all other nodes. In a decentralized network, multiple locally central nodes manage communication, but each node only communicates with the locally central node it is connected to. In a distributed network, every node has the capability to communicate with any other node in the network, with no single point of control or failure. Decentralization, in Baran's context, was a step away from the fragility of centralized systems, offering improved resilience since the system could continue to operate even if one node failed. However, true distribution was the ideal, as it offered

the most robust solution: a network with no single points of failure, capable of rerouting communications via multiple paths in the case of a node or connection failure, thereby ensuring the network's integrity and continuous operation even under adverse conditions.

A significant contribution to the field was the development of the ARPANET in the late 1960s, which would later evolve into the modern Internet [7]. Its design was rooted in the visionary concept of a globally interconnected set of computers, allowing users to access data and programs from any site. The main philosophy behind ARPANET was the decentralization of control and the use of packet switching. The development of ARPANET faced several challenges, including the need for a reliable end-to-end protocol to maintain effective communication in the face of potential interference and creation of algorithms to prevent lost packets from permanently disabling communications. The ARPANET's design principles of resilience, decentralization, and open architecture have had a lasting impact, influencing subsequent generations of network design.

In decentralized systems, data consistency issues arise due to inherent challenges in maintaining a uniform state across multiple independent nodes or processes, especially in the face of updates occurring simultaneously at different locations. This problem is exacerbated by factors such as network latency, partitioning, and node failures, which can lead to conflicting states within the system [8]. Ensuring that all nodes in a decentralized system agree on a single data state or sequence of events—known as achieving consensus—is critical for the system's reliability and trustworthiness. However, reaching consensus is non-trivial, particularly when system components might fail or behave maliciously. This problem is famously encapsulated in the Byzantine Generals' Problem, a term coined by Leslie Lamport [9]. The problem describes a situation where several generals, each commanding a portion of the Byzantine army, prepare to attack a city. They need to coordinate their attack plan but are located at different camps and can only communicate via messengers. Some of these generals might be traitors, sending false messages or no messages at all, intending to disrupt the coordinated attack. The challenge lies in designing an algorithm that allows the loyal generals to agree on a common battle plan, ensuring that all loyal generals either attack or retreat, even in the presence of traitorous generals who may be sending conflicting information. The Byzantine Generals' Problem is a metaphor for the real-world challenge of achieving consensus in a distributed or decentralized system, where components might not only fail but could also be actively deceptive. Solutions to this problem, known as Byzantine Fault Tolerance (BFT) algorithms [10], are fundamental in many modern decentralized systems to ensure that the system can handle not just technical faults but also adversarial behavior.

The introduction of peer-to-peer (P2P) networks in the late 1990s and early 2000s, initiated by platforms like Napster, marked a departure from traditional client-server models, promoting direct user interactions but revealing centralization flaws, as seen in Napster's reliance on a central server [11]. Subsequent platforms like Gnutella and Freenet [12] introduced decentralized elements, such as query flooding and encrypted file snippets, improving security and anonymity but often sacrificing efficiency. BitTorrent [13] further advanced P2P technology with its distributed hash table (DHT) [14], enabling decentralized indexing and peer discovery. It has become evident that peer-to-peer architecture also brings a number of non-technical challenges [15]. First, it is difficult to ensure content integrity without a centralized control mechanism, leading to the potential for malicious or

corrupted files to be distributed. Second, the architecture relies on cooperation, with selfish peers impacting the system's performance and reliability. These complexities underscore the multifaceted nature of P2P systems, reflecting both their potential and the challenges they must navigate.

Bitcoin [1], introduced in 2008 by the pseudonymous entity Satoshi Nakamoto, represents a groundbreaking application of decentralized technology, offering a digital currency solution that operates independently of central financial institutions. It's built on blockchain technology, a distributed ledger system that records transactions across many computers so that the record cannot be altered retroactively. This innovation addressed key vulnerabilities identified in previous decentralized systems, such as content integrity in P2P networks, by introducing cryptographic security measures and a consensus mechanism known as proof of work [16]. What sets Bitcoin apart is its ability to mitigate issues of trust, content integrity, and operational resilience that were prevalent in earlier systems. Unlike the P2P file-sharing systems of the past, Bitcoin ensures data integrity through a publicly verifiable ledger and incentivizes user cooperation through mining rewards.

The advent of Bitcoin and other blockchain systems sparked renewed interest in decentralized technologies and became the precursor to what is now referred to as Web3. This movement extends beyond cryptocurrency, influencing the development of decentralized applications (DApps), decentralized autonomous organizations (DAOs), and more.

1.1.2 Evolution of the Web

The World Wide Web [17] has undergone profound transformations since its inception, evolving in response to technological advancements, societal needs, and user behaviors. As a complex, multifaceted ecosystem, the web's development can be analyzed through various lenses. However, three aspects stand out due to their significant impact on user experience and the structural evolution of the internet: *control over data, economic model*, and *governance*. These aspects are pivotal as they encapsulate the data autonomy, economic implications, and power dynamics that define each phase of the web's evolution. Control over data is crucial in understanding user autonomy and privacy, indicating who holds the information, who can monetize it, and who decides on its usage. The economic model reflects the mechanisms through which platforms generate revenue, highlighting the underlying incentives and how they affect users. Governance determines the processes through which decisions are made and identifies the entities responsible for making these decisions. By examining these three aspects, we critically assess the web's progression from a static information repository to a dynamic, user-generated content space, and towards its potential future as a decentralized ecosystem. In this context, Table 1.1 provides a succinct comparison of the Web generations, namely Web1, Web2, and Web3, highlighting the paradigm shifts in control over data, economic models, and development and governance structures.

The initial phase of the internet, known as Web1.0, or the "Static Web", was characterized by informational websites that were read-only [18]. Content creation was an arduous task, typically reserved for individuals or organizations with technical expertise. The technology underpinning this phase was dominated by simple HTML used to render static web pages. Interactivity with [19] and within these websites was minimal, akin to online brochures. Control over data during this phase was centralized, residing solely with website owners.

Aspect	Web1	Web2	Web3
Control over Data	Centralized (Website Owners)	Centralized (Platforms)	Decentralized (Users)
Economic Model	Transactional	Advertisement-driven	Tokenized
Governance	Centralized	Centralized	Decentralized

Table 1.1: An overview of the evolution of the Web, categorizing the transformation across the three primary aspects: Control over Data, Economic Model, and Governance.

The economic model was straightforward and transactional, often based on the sale of goods or services or on subscription models. Development and governance were also centralized, with decisions made by website developers and owners with little to no input from end-users.

Web2, known as the 'Social Web', transitioned to a more dynamic architecture, leveraging technologies like AJAX [20] to enable real-time collaboration and user-generated content. Platforms like Facebook, YouTube, and Twitter thrived, focusing on user engagement and continuous content creation. However, control over data remained centralized with platform owners despite the increased interactivity, leading to privacy and security concerns [21]. The economic model evolved to become advertisement-driven, giving rise to the "attention economy," where user attention became a valuable commodity. While development continued to be centralized, there was an element of user feedback that influenced platform modifications and content curation.

Web3.0, initially conceptualized by Tim Berners-Lee as the "Semantic Web" [22], aimed to create a web of data with discernible meaning, utilizing formats like the Resource Description Framework (RDF). Nonetheless, the concept has since morphed, with "Web3" now synonymous with the "Decentralized Web." This new paradigm emphasizes privacy and security, advocating peer-to-peer internet exchange without intermediaries. Here, control over data is decentralized, giving users autonomy over their information, facilitated by technologies like blockchain and data encryption. The economic model is anticipated to shift towards tokenization, where cryptocurrencies and tokens facilitate transactions and incentivize various activities [2]. Development and governance in Web3 aspire to be decentralized, potentially utilizing structures like decentralized autonomous organizations (DAOs) for collective decision-making [23]. As of this writing, however, Web3 remains a developing concept, with ongoing discussions about its characteristics and functionalities.

1.1.3 Design Philosophies of Web3

The protocols and systems underpinning Web3 are guided by a set of design philosophies integral to shaping their functionality and user experience. Central to these philosophies is a commitment to decentralization, a guiding principle that pervades every aspect of Web3's architecture. These philosophies are rooted in the concept of redistributing power from centralized entities to network participants, fostering a more equitable and user-empowered digital ecosystem.

1

Layered Architecture

Layered architecture is a fundamental design principle in Web3, pivotal for extending the system's capabilities across various functionalities efficiently. This architectural approach divides the system into distinct layers, each dedicated to specific tasks, yet interconnected to form a cohesive ecosystem.

Layers are not a novel and unique to Web3. For example, OSI [24] uses layers to standardize network communications across different hardware. OSI model is built on a rigid, hierarchical structure where each layer has specific, well-defined responsibilities, and interactions between layers are strictly regulated. This structured approach aids in diagnosing network issues and ensures consistent behavior across different implementations.

The OSI model is underpinned by a comprehensive set of standards, established by organizations like the ISO (International Organization for Standardization) and the IETF (Internet Engineering Task Force). These standards ensure that devices and applications from different vendors can communicate effectively over the network. Web3, on the other hand, values interoperability operating in a more less standardized environment. Interoperability in Web3 is achieved through a combination of open protocols, community-driven standards, and bridging technologies, rather than through rigid adherence to a universal model. To the moment, a typical way to divide Web3 into two layers: the Layer-1, which handles fundamental blockchain operations such as consensus mechanisms and data storage, and Layer-2, which introduces additional functionalities like off-chain transaction processing, privacy enhancements, and interoperability solutions.

Specific systems in Web3 demonstrate the efficacy of this layered approach. Taking Ethereum as an example, its base layer functions as a decentralized ledger facilitating smart contracts and DApps interactions. On top of this, Layer-2 solutions like Optimism [25] and Arbitrum [26] use Optimistic Rollups to execute transactions off the main chain, thereby enhancing scalability. Similarly, Bitcoin's blockchain serves as its own Layer-1, providing a secure and decentralized ledger for cryptocurrency transactions. Building upon this, the Lightning Network [27] operates as a Layer-2 solution, enabling low-cost Bitcoin transactions ideal for micropayments.

Transparency and Trust-Minimization

Transparency in Web3 refers to the visibility of the operations on the network. Every participant has access to the public ledger, ensuring that actions within the network are openly verifiable. This concept fosters trust in the system, ensuring users can rely not only on the outputs but also on the processes leading to those outputs. Transparency is closely linked with trust minimization, a concept that aims to reduce reliance on central authorities, instead embedding trust within the technology and its decentralized processes. Transparency and trust minimization manifest themselves in various ways, including the verifiability of blockchain transactions, self-sovereignty, open-source development, and community-based governance models.

Blockchain verifiability ensures that all network transactions are verifiable and only valid data is accepted into the ledger, which is essential for Web3 transparency. The validity of transactions is determined by their traceability to their origin and adherence to application invariants, such as maintaining positive wallet balances or ensuring consistent contract logic. Blocks containing invalid transactions are not accepted. In principle, any

node can replicate all blocks and transactions to verify the blockchain's validity themselves. However, in practice, most users utilize public block explorers, such as Etherscan [28] and Blockchain Explorer [29]. These services enable users to verify transaction details and the correctness of smart contract interactions.

Self-sovereignty implies that users have complete authority and control over their digital assets and identity. Users interact with the network through *wallet* clients, which play a crucial role in preserving user autonomy and privacy. These wallets securely store private keys—the critical element for asset control—and facilitate user interactions with the blockchain. Users sign transactions using their private keys, ensuring that only they can authorize the transfer or modification of their assets. Once signed, these transactions are broadcast to the network and eventually added to the blockchain. Wallet clients can later verify that transactions are finalized.

Open-source development is a pivotal aspect of Web3. This approach involves making the source code of projects publicly accessible, allowing anyone to contribute, audit, or adapt the code. This openness enables the community to audit security and ensure alignment with user needs and industry standards. Additionally, it fosters a community-driven approach to innovation, resulting in numerous forks. Examples include Ethereum Classic, which emerged following a divergence in the Ethereum community, Litecoin as a lighter version of Bitcoin, and various Bitcoin hard forks like Bitcoin Cash and Bitcoin SV [30].

Decisions in the Web3 space, particularly those affecting protocols and key system updates, are made in a transparent manner. Web3 projects often employ decentralized governance models, where token holders or community members have a say in key decisions. Governance tokens grant voting rights, enabling token holders to participate in proposals and decision-making processes. This type of governance is called a *Decentralized Autonomous Organization (DAO)*. DAOs represent collectives governed by their members, where decisions are made through proposals and voting mechanisms encoded in smart contracts.

Tokenization and Economic Incentives

Economic incentives are a third cornerstone in Web3. They serve as key mechanisms for encouraging network participation, governance, and value exchange. Typically implemented as tokens, these incentives are embodied in various forms across multiple protocols. Bitcoin pioneered the concept of cryptocurrency. Beyond its use as a medium of exchange, Bitcoin also serves as the means to pay transaction fees and reward miners. A key aspect of Bitcoin's design is its limited total supply, a feature that instills scarcity and potentially drives its value up over time, thereby incentivizing long-term network participation and investment.

Subsequent blockchain systems have either copied or expanded upon Bitcoin's incentive mechanism. For example, Ethereum, with its native token Ether, compensates for computational efforts in executing smart contracts. Ethereum's introduction of smart contracts added a new dimension to the utility of blockchain tokens, enabling a wide array of decentralized applications. Decentralized finance platforms like Compound [31] (COMP) and Aave [32] (AAVE) have introduced governance tokens, offering users a say in protocol development and decision-making processes. Projects like Filecoin [33] and Golem [34] have innovatively utilized utility tokens, FIL and GNT respectively, to create decentralized

1

marketplaces for computational resources. In Filecoin, FIL tokens incentivize users to offer their unused storage space, building a distributed file storage network. Golem, with its GNT tokens, similarly rewards users who contribute their computing power, supporting a decentralized computational resource pool.

1.2 Web3 Foundations

This section delves into the foundational elements that underpin Web3. At its core, Web3 is built on the concept of a distributed ledger, predominantly realized through blockchain technology. We will discuss the various design choices and trade-offs inherent to these ledger systems, highlighting the differing assumptions about trust that they entail.

In addition to the distributed ledger, we cover the concept of the 'Web3 Stack'. This term encapsulates the myriad layers and technologies that, together, facilitate the operation of decentralized applications (DApps). Through this exploration, we aim to provide a comprehensive understanding of the key components that constitute the foundation of Web3.

1.2.1 Blockchain

Transactions in the Bitcoin network are signed by users and submitted to network peers. A special set of peers, called miners, participate in the consensus process. Each miner periodically batches received transactions into a block, selects the last previous block in the locally known longest chain, and attempts to append the new block by solving a cryptographic puzzle. This process, known as mining, requires finding a value that, when hashed with the block's contents, produces a hash that meets certain criteria (e.g., having a specific number of leading zeros). The first miner to solve the puzzle broadcasts the solution to the network, and other miners verify it. Upon verification, the new block is added to the blockchain, and the miner who solved the puzzle is rewarded with newly minted Bitcoins and transaction fees.

The design of the Bitcoin system has profoundly influenced the architecture of subsequent blockchains. The consensus algorithm inspired by Bitcoin, colloquially known as 'Longest-Chain-Wins', is usually referred to as Nakamoto consensus [35]. These blockchain systems are categorized as *permissionless* networks because they do not rely on specific permission restrictions to operate correctly, in contrast to permissioned networks. In permissionless networks, anyone can join the network, participate in the consensus process, and validate transactions without needing approval from a central authority.

As of this writing, there are over 1,000 live blockchain systems, most of which are inspired by Nakamoto's design. Initially conceived to underpin cryptocurrencies, this technology has found applications beyond financial transactions through the advent of smart contracts and DApps. Ethereum is the most notable example in this regard [2].

However, the proliferation of permissionless blockchains has also exposed inherent design limitations. Specifically, the Proof-of-Work (PoW) consensus algorithm has demonstrated limited throughput, both in practice and theoretical analysis [36]. This realization has boosted the exploration of alternative, scalable architectures.

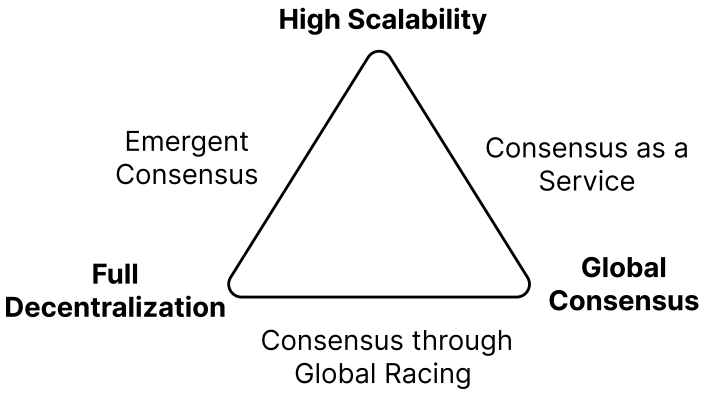


Figure 1.1: The Blockchain Triangle suggests that it is possible to achieve only two of the three common goals - High Scalability, Full Decentralization, and Global Consensus. The sides on the triangles represent consensus families that sacrifice one of the goals.

1.2.2 Design Choices of Distributed Ledgers

Distributed ledgers serve as a more generalized concept than Bitcoin's blockchain, offering a broader range of applications beyond digital currencies. These ledgers function as databases that are distributed across a multitude of nodes. Each node is responsible for maintaining an individual copy of the ledger, thereby ensuring both transparency and fault tolerance within the system.

One of the most critical challenges that these systems aim to address is the issue of consistency, famously exemplified by the 'double-spending' problem [1]. In digital currencies, double-spending refers to the fraudulent act of spending the same digital token multiple times. For example, in a widely distributed network, a malicious actor could duplicate their digital tokens and use them in transactions with different parties, leading to a *conflict* in the transaction history. To resolve such conflicts, distributed ledger systems employ consensus algorithms. These algorithms resolve data discrepancies by ensuring that only one, agreed-upon version of the transaction record is maintained across all nodes.

Blockchain Trilemma

Given the significant overhead commonly linked to consensus algorithms, the structural design of blockchain and distributed ledger systems is essential. These systems encounter various trade-offs among performance, security, and decentralization. To frame these trade-offs, the concept of the 'Blockchain Trilemma' has been proposed [37], similar to the CAP theorem in distributed systems [38]. This trilemma posits that simultaneously achieving scalability, decentralization, and security is a significant challenge. For this discussion, the term "security" is replaced with the more specific concept of "global consensus," as illustrated in Figure 1.1. Ledgers are categorized into three types based on the goals they prioritize.

Consensus through Global Racing systems prioritize complete decentralization and achieve global consensus through periodic leader selection via methods like lotteries or

1

races in defined rounds. Rounds refer to the cycles or phases through which the network nodes undergo to reach agreement or consensus on the current state of the ledger. During each round, nodes compete to propose a new block to be added to the chain.

However, this mechanism imposes significant constraints on the frequency of new block creation. Nodes need to wait for one or multiple rounds to determine the leader with reasonable certainty. The requirement for multiple rounds before achieving consensus on a new block leader ensures fairness and reduces the risk of fraudulent activities, such as double-spending attacks and selfish mining [39, 40].

Selfish mining is a strategy where a miner or group of miners keeps their discovered blocks private instead of broadcasting them to the entire network. By selectively revealing their blocks, selfish miners aim to waste other miners' computational resources on already solved puzzles, thus increasing their own chances of earning rewards. This can lead to forks in the blockchain, as different parts of the network may work on different versions of the blockchain history, thereby undermining the security and consensus of the system.

This requirement to wait for multiple blocks combined with the fixed average block time (about 10 minutes for Bitcoin) and a 1 MB block size limit, restricts the number of transactions that can be processed per second. Attempts to tune this problem towards higher throughput by not waiting for multiple blocks to be appended, e.g., the zero-confirmation of Bitcoin Cash, have a much higher rate of forks [41, 42].

Consensus as a Service[43] systems emphasize scalability and employ traditional consensus algorithms[44], which are inherently limited in their ability to scale due to the message complexity involved in coordination. This necessitates severely restricting the number of nodes actively participating in the quorum, thereby sacrificing full decentralization. However, if decentralization is not a goal and fully trusting a small set of privileged entities is acceptable, blockchains offer little benefit over traditional systems of record. Examples of such systems include Ripple [45], Hyperledger Fabric [46], and EOS [47].

The third category, termed *Emergent Consensus*, encompasses systems that aim to reconcile full decentralization with high scalability. These systems employ conflict-resolution mechanisms and adhere to the principles of *strong eventual consistency*, thereby forgoing immediate global consensus. For example, IOTA utilizes a Directed Acyclic Graph (DAG) structure known as the Tangle, diverging from the traditional blockchain model [48]. Trustchain [49], another exemplar, features a personalized hash chain of transactions for each user. Additionally, sharding techniques [50] and Layer-two solutions like the Lightning Network [27] and Plasma [51], although not fully autonomous, employ similar design philosophies. These approaches streamline transaction processing by limiting the number of involved parties, often reducing them to just two parties.

Trust Spectrum

To provide a comprehensive understanding of the varied landscape of distributed ledger systems, we introduce a conceptual framework illustrating the connection between trust assumptions and consensus algorithms. This relationship is depicted as a spectrum in Figure 1.2, extending from left to right, starting with complete trust in a single entity and progressing towards minimal or no trust in any individual party.

At the far left of the spectrum are *centralized* systems, such as traditional banking systems and centralized digital asset exchanges like Coinbase [52]. These systems, controlled

Figure 1.2: Spectrum of distributed ledger designs with their trust assumptions, ranging from full trust to minimal trust (Centralized to Algorithmic Lottery).

by a single entity, require users to place full trust in them for data consistency and validity. The centralized nature of these systems often results in efficient processing but at the cost of transparency and user autonomy.

Next to centralized systems are *permissioned* blockchains, which rely on *static committees* for governance. Examples include platforms like Hyperledger Fabric [46]. These systems introduce a higher degree of decentralization but still necessitate substantial trust in a selected group of nodes. These nodes, often predetermined and invited, have the exclusive right to validate transactions and create new blocks. While these systems offer increased transparency compared to fully centralized systems, they remain limited in terms of broad-based participation.

A step away from static committees, we encounter systems that operate on a *federation* model, such as Stellar [53] and Ripple [45]. These platforms employ Federated Byzantine Agreement (FBA) for consensus, enabling decentralized control without requiring all nodes to be known or verified in advance. For example, Ripple employs a Unique Node List (UNL) of validators recognized to participate in the consensus protocol. The network does not enforce a single UNL; however, Ripple and the XRP Ledger Foundation maintain the default UNL, introducing potential centralization risks.

The *dynamic committee* category includes systems that utilize consensus mechanisms like Proof-of-Stake (PoS)[54]. In PoS, validators are chosen based on criteria such as token holdings used as collateral, called a *stake*. Validators are selected to form a committee that runs a consensus algorithm. Typically, the probability of being selected as a validator depends on the total amount of tokens reserved as a stake. While PoS systems strive for a balance between decentralization and efficiency, they must be meticulously designed to prevent wealth concentration from influencing the neutrality of the consensus processes[55].

At the rightmost end of the spectrum are systems based on an *algorithmic lottery*, such as Bitcoin's Proof-of-Work (PoW) model. These systems are open to any participant willing to commit resources, such as computational power in PoW, to maintain network integrity. The use of cryptographic techniques ensures fairness and randomness in node selection. However, this openness often comes with challenges in scalability and energy efficiency.

1.2.3 Web3 Stack

The Web3 stack, as illustrated in Figure 1.3, is a structured representation of the various layers that constitute the Web3 ecosystem. This stack is divided into four layers: Access, Application, Protocol, and Infrastructure. For each layer, we present four examples of associated technologies. Note that our Web3 stack differs from the one discussed in Section 1.1.3, as it incorporates a wider range of technologies, including both 'Layer-1' and

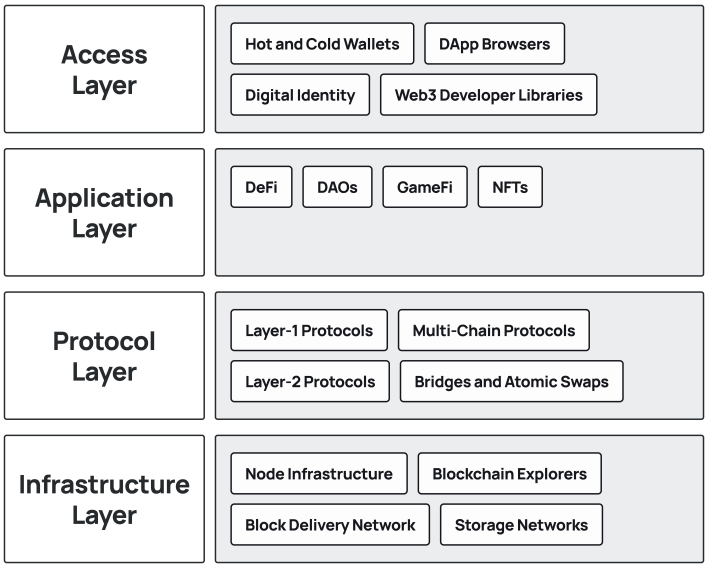


Figure 1.3: The Web3 stack consists of four layers: Access, Application, Protocol, and Infrastructure. For each layer, we provide four prominent example technologies.

'Layer-2', as part of the Protocol Layer of the Web3 stack.

Infrastructure Layer

The Infrastructure Layer forms the foundational base of the Web3 stack, offering essential services and technologies crucial for the operation of all subsequent layers. This layer represents technologies to enable blockchain as a system: providing node hardware for the network, relay networks to deliver the messages, data explorers to the blockchain networks, and storage networks.

Central to this layer is the *node infrastructure* that underpins the blockchain network. This includes various devices running blockchain software, enabling network participation. Nodes typically fall into two categories: full nodes, which store the entire blockchain and partake in the consensus process, and light nodes, which submit transactions and verify their status without maintaining a complete copy of the blockchain. Depending on the network, there might be other specialized roles. For example, mining nodes in Bitcoin run the Proof of Work (PoW) algorithm, while in Ethereum 2.0, proposer and builder nodes [56] focus on delivering the content of the block and creating the block, respectively.

Message delivery networks are essential components that enhance the efficiency of data propagation in blockchain systems. Block delivery networks like BloXroute [57] optimize the propagation of blocks across the network, thereby reducing latency and increasing the overall throughput. These networks employ advanced routing protocols and compression techniques to ensure efficient data transmission, even under high network load conditions. Transaction delivery protocols such as RLPx [58] or Waku [59] further streamline the

process by facilitating the quick and reliable dissemination of transaction data across the network.

Running a full node requires significant resources, such as storage, due to the need to store all blocks of the blockchain. As a result, centralized services often fill this gap, providing the necessary infrastructure to access blockchain data. Services like Infura [60], Alchemy [61], and QuickNode [62] provide APIs to enable easy access to blockchain data without the need to run a full node. Moreover, *block explorers*, such as Etherscan [28] are centralized services that provide users access to the transaction status without the need to run any light node.

To mitigate reliance on centralized systems, the concept of decentralized infrastructure providers has emerged. These entities operate within a decentralized network, delivering the computational and storage capabilities that DApps frequently necessitate. *Storage networks*, such as InterPlanetary File System (IPFS) [63] is a prime example of a decentralized infrastructure provider. IPFS introduces a peer-to-peer protocol engineered for the distributed storage and exchange of hypermedia across a decentralized file system. By dispersing files among numerous nodes, IPFS bolsters network resilience by retrieving data from the nearest available node. Within the Ethereum framework, IPFS has gained traction as an cost-effective solution for managing media that is either referenced by or utilized within a smart contract.

Protocol Layer

The Protocol Layer in the Web3 stack encompasses the technologies that define how data is processed, validated, and agreed upon across the blockchain network. It's named the "Protocol Layer" because it consists of the essential protocols—agreed-upon methods and rules—that facilitate communication and consensus among distributed nodes, both within a single blockchain and across multiple different blockchain networks. On this level we include both 'Layer-1' and 'Layer-2' blockchains and inter-blockchain protocols, such as Multi-chain, bridges and swaps.

Layer-1 blockchains are the primary networks that handle all on-chain transactions. They form the main ledger where transactions are recorded and validated. Examples include Bitcoin, which uses Proof-of-Work (PoW) to secure its network, and Ethereum, which is transitioning to Proof-of-Stake (PoS). These blockchains provide the basic ledger and consensus mechanisms essential for maintaining the integrity and security of the decentralized network.

To overcome the scalability limitations of Layer-1 blockchains, Layer-2 protocols are developed to process transactions off-chain. Solutions like the Lightning Network [27] for Bitcoin and Plasma [51] for Ethereum allow for faster and cheaper transactions by handling them outside the main blockchain and only settling the final results on-chain. This approach reduces congestion and increases the transaction throughput of the primary network. Typically, Layer-2 protocols operate as simplified or specialized distributed ledgers that often involve smart contracts used on Layer-1 or special nodes that can handle a large volume of transactions quickly and efficiently. The final outcomes of these transactions are then consolidated and recorded on the Layer-1 blockchain.

As the Web3 ecosystem consists of numerous blockchain networks, interoperability becomes a key concern. There are several approaches to facilitate the transfer of assets and information between different blockchain systems:

1

Atomic swaps are a method for exchanging cryptocurrencies between users on different blockchains without the need for an intermediary. They use smart contracts to ensure that the exchange either happens entirely or not at all, minimizing the risk of one party defaulting. For example, an atomic swap allows a user to trade Bitcoin for Ethereum directly with another user, ensuring that both parties receive their assets securely and simultaneously.

Bridges are protocols that enable the transfer of tokens and data between different blockchain networks. They typically involve locking tokens on the source blockchain and minting equivalent tokens on the destination blockchain. For example, a bridge can allow users to wrap Bitcoin (creating Wrapped Bitcoin, or WBTC) to be used on the Ethereum network. This enables Bitcoin holders to participate in Ethereum-based DeFi applications while keeping their original Bitcoin assets secure.

Multi-chain protocols aim to create an ecosystem of interoperable blockchains that can communicate and share data seamlessly. Unlike atomic swaps and bridges, which facilitate interoperability on an ad-hoc basis, multi-chain protocols are designed to support interoperability at a fundamental level. Examples include Cosmos [64] and Avalanche [65]. Cosmos uses the Inter-Blockchain Communication (IBC) protocol to enable different blockchains to transfer data and assets between each other. Avalanche allows multiple subnets (customized blockchains) to interoperate within its ecosystem, providing flexibility and scalability.

Application Layer

The Application Layer are end-user applications and services. This layer leverages the underlying infrastructure and protocol layers to deliver practical functionalities directly to users, making blockchain technology usable in everyday scenarios. This layer includes a diverse range of applications such as Decentralized Finance (DeFi), Decentralized Autonomous Organizations (DAOs), Non-Fungible Tokens (NFTs), and GameFi platforms.

DeFi represents the implementation of financial services within Web3, operating without traditional intermediaries. Using smart contracts, DeFi protocols facilitate activities such as lending, borrowing, and trading of assets. Examples include Uniswap [66], a decentralized exchange (DEX) on the Ethereum blockchain, and Compound [31], which enables peer-to-peer borrowing and lending. Research in DeFi focuses on improving efficiency, security, and scalability, addressing challenges like liquidity provision, impermanent loss, and integration with traditional finance. Stablecoins [67] are another significant area, engineered to maintain stable value by pegging to more stable assets.

DAOs offer a novel approach to organizational management and decision-making. These organizations use smart contracts to automate governance processes, allowing stakeholders to propose, deliberate, and vote on decisions without centralized intermediaries. Research in this area is aimed at developing effective governance models and voting mechanisms [68]. Various frameworks include token-based voting, where decision-making power correlates with the number of tokens held [69], and reputation-based systems [70], where votes are weighted based on contributions or expertise within the community.

NFTs are digital assets representing ownership and provenance of unique items or content. Each NFT is distinct, with unique value due to its characteristics and rarity. This

uniqueness and the blockchain's immutable nature make NFTs suitable for representing digital art, collectibles, virtual real estate, and more. Platforms like OpenSea [71] facilitate NFT trading. Current research focuses on enhancing usability, privacy, and standardization of NFTs [72], ensuring seamless integration into digital ecosystems.

GameFi integrates financial incentives into the gaming industry, offering in-game assets, tokens, or other rewards with real-world value. Games like Axie Infinity [73] and Decentraland [74] incorporate GameFi elements, allowing players to earn income through gameplay, trading in-game items, or participating in the game's governance [75]. These games create new economic opportunities and a symbiotic relationship between gaming and finance.

Access Layer

The Access Layer acts as the primary interface for users to engage with Web3 technologies. This layer includes digital wallets, identity management solutions, blockchain-enabled browsers, and development libraries, all designed to make blockchain technology accessible and user-friendly.

Hot wallets, such as MetaMask [76], operate online and offer convenient, quick access to funds, making them suitable for regular transactions. They are typically browser-based or mobile applications that allow users to send, receive, and manage digital assets easily. However, because they are connected to the internet, they are more vulnerable to hacking and other threats. *Cold wallets*, such as the Ledger Nano [77], are offline storage solutions that provide enhanced security for asset storage. They are hardware devices that store private keys offline, making them ideal for long-term holdings or large amounts of digital assets. The choice between hot and cold wallets involves balancing convenience against security, depending on user needs. Hot wallets are great for everyday use, while cold wallets are better suited for securing larger sums of cryptocurrencies over extended periods.

Using the same wallet clients, individuals can manage their *digital identities*, ensuring that they are the sole arbiters of their personal information. Projects like uPort [78] and Civic [79] exemplify this paradigm, allowing users to own and control their identity without the need for centralized authorities. This approach is referred to as self-sovereign identity [80]. Unlike traditional identity systems, which are often siloed and controlled by central entities, self-sovereign identity is based on the principle that individuals have the right to own, control, and present their identity information as they see fit. This approach leverages blockchain technology to create a decentralized and secure framework for identity management, where trust is established through cryptographic proofs rather than centralized validation.

Browsers like Brave [81] and Opera's Crypto Browser [82] serve as gateways to decentralized applications. They integrate blockchain connectivity and wallet functionality, offering a seamless experience for users to interact with DApps. These browsers often come with built-in security features and are designed to facilitate a more private and decentralized browsing experience. Brave, for instance, integrates its native cryptocurrency, Basic Attention Token (BAT), within its advertising system, offering an alternative monetization model for web content creators. Users can earn BAT by opting into privacy-respecting ads, which they can then use to tip their favorite websites and content creators.

Much like browsers help users engage with DApps, libraries such as Web3.js [83] serve as tools for developers in the Web3 space. Web3.js is a collection of JavaScript libraries

that enable seamless interaction with a local or remote Ethereum node via an HTTP or IPC connection. This toolset is designed to expedite the development process, allowing developers to focus on the core functionality of their DApps without getting entangled in the underlying blockchain complexities. It provides the necessary functions to interact with smart contracts, make blockchain data queries, and subscribe to events happening on the blockchain. Web3.js abstracts the complexities of Ethereum's RPC (Remote Procedure Call) protocol, providing developers with simple methods to interact with the Ethereum network.

1.3 What Web3 Lacks

As with any evolving technology, Web3 encounters its own set of challenges and limitations. Despite its promise of decentralization, enhanced security, and user empowerment, Web3 still faces significant hurdles that hinder its widespread adoption and functionality. Among the numerous properties and limitations in the current Web3 landscape, some of the most popularly mentioned are scalability issues, high energy consumption, complex user interfaces, and regulatory uncertainties. Scalability remains a major concern, as many blockchain networks struggle to handle a high volume of transactions without compromising speed and efficiency. High energy consumption, particularly in Proof-of-Work (PoW) systems like Bitcoin, raises environmental and sustainability concerns. Complex user interfaces can deter non-technical users from engaging with Web3 applications, while regulatory uncertainties pose challenges to legal compliance and the integration of blockchain technologies into existing frameworks.

Although there are many areas that require improvement, we focus on four of the most prominent and specific challenges: reliability under real-world conditions, accountability of network nodes, resilience to Sybil attacks, and incentive alignment of the tokenomics models. These specific issues have been chosen because they are fundamental to the core functioning and trustworthiness of Web3 systems. Addressing these challenges is crucial for enhancing the overall robustness, security, and economic viability of decentralized networks. By understanding and tackling these shortcomings, we can advance Web3 and address these weaknesses.

1.3.1 Reliability

Reliability in distributed systems is defined as the ability to produce correct output within a given time [84]. The appeal of decentralized systems, like blockchain, lies in their potential for high reliability despite the inherent challenges of distributed networks. However, delays in transaction verification, block propagation, or consensus can result in blockchain forks and inconsistencies. Therefore, as blockchain adoption grows, its performance especially under high loads becomes a crucial factor in its reliability.

Transaction throughput, often reported in transactions per second (TPS), is a commonly used metric for blockchain performance. However, this metric can be misleading. Many projects report theoretical maximums achieved under idealized conditions, which may not be representative of real-world performance. For example, a blockchain network might claim a high TPS based on tests conducted in a controlled environment with few nodes, ignoring network latency.

One notable instance highlighting the discrepancy between theoretical and actual performance occurred in December 2017, when the Ethereum network experienced severe congestion due to the unexpected popularity of a DApp called CryptoKitties [85]. The game accounted for approximately 11% of all transactions on the Ethereum network at its peak. Despite Ethereum's theoretical throughput of around 30 TPS, the actual throughput was much lower during this period, leading to delayed transactions and increased gas fees. This incident underscored the importance of considering real-world conditions when evaluating blockchain performance. Network congestion not only affects transaction speeds but also increases transaction costs, as users must pay higher fees to prioritize their transactions. This can make blockchain applications impractical for high-frequency or micro-transactions. Furthermore, the reliability of a blockchain under high load conditions is crucial for its adoption in mainstream applications where consistent and timely performance is expected.

Another common performance metric, latency, refers to the time it takes for a transaction to be added to the blockchain and finalized. Latency can vary significantly during periods of high demand, complicating the assessment of a blockchain's reliability. In the Web3 ecosystem, decentralized exchanges (DEXs) often experience high latency during periods of intense trading activity. For instance, during significant market movements, users may find that their transactions take an unusually long time to be confirmed, or they may even fail altogether. This latency undermines the utility of DEXs for time-sensitive trading strategies and raises questions about the network's reliability under stress. High latency can lead to slippage, where the execution price differs from the intended price, which can be detrimental to traders relying on precise timing.

Scalability refers to a blockchain's ability to maintain performance in terms of average throughput and latency as the number of participating nodes increases. A blockchain system might include a large number of nodes participating in the consensus process; for example, at the time of writing, Bitcoin has around 17,000 nodes, and Ethereum has around 8,000 nodes. Scalability is a critical factor in assessing a blockchain's long-term reliability, as it involves the ability to successfully include more nodes without compromising the system's liveness or security.

The Bitcoin network faced a significant scalability issue that culminated in the Bitcoin Cash [86] hard fork in 2017. The debate centered around Bitcoin's 1MB block size limit, which constrained the network to approximately seven transactions per second. As the network grew, this limit led to slower confirmation times and higher transaction fees, affecting the system's reliability. The inability to process a higher volume of transactions efficiently led to congestion, making the network less reliable for users who needed quick transaction processing. The Bitcoin Cash hard fork increased the block size to 8MB, aiming to provide a more scalable solution. This event highlighted the importance of scalability in maintaining a blockchain's usability and reliability as adoption grows.

These examples underscore the complexity of assessing blockchain reliability. While metrics like transactions per second (TPS), latency, and scalability provide valuable insights, they are not sufficient when reported in isolation. Real-world events, such as sudden spikes in demand or contentious community debates, can significantly impact these metrics. During periods of high demand, blockchain networks typically experience increased transaction fees, extended confirmation times, and decreased system throughput.

Therefore, a more nuanced understanding is required, considering both theoretical

capabilities and empirical performance under varying network conditions. Evaluating reliability should involve stress testing under different scenarios, monitoring the network during peak usage times, and understanding the impacts of proposed changes or upgrades. Additionally, examining how well a blockchain can recover from disruptions, such as forks or attacks, is essential in assessing its overall reliability.

1.3.2 Accountability

Accountability in distributed systems [87] refers to the ability to cryptographically link all actions to the nodes performing them, with a secure record of past actions being maintained. This enables nodes to verify each other's actions for correctness and ensures that any fault observable by a correct node is eventually detected, providing irrefutable evidence linked to the faulty node. In blockchain, accountability involves detecting and proving malicious actions by nodes, such as transaction or block censorship, unfair ordering, and other protocol deviations. A key challenge in blockchains is achieving accountability without introducing new trust assumptions.

One of the most salient challenges to accountability in blockchain in recent years is the emergence of Miner Extractable Value (MEV) [88]. MEV refers to the potential profit a nodes can make by strategically including, excluding, or reordering transactions within the blocks they produce. Initially, blockchain systems primarily incentivized participants through block rewards, designed to ensure that following the protocol was the most profitable strategy. However, MEV introduces an additional incentive that allows validators to manipulate transaction ordering for profit, challenging this initial assumption.

Nodes can prioritize their own transactions or those of the highest bidders, leading to concerns about fairness and system stability. This behavior deviates from the expected protocol-prescribed behaviors and introduces a new layer of complexity in maintaining accountability. While MEV can serve as an additional revenue stream for miners, it raises significant concerns about fairness and the overall stability of the blockchain network [89].

Another pressing issue is transaction censorship by validators. There is empirical evidence [90] that substantiates the real-world implications of this issue, particularly in the Ethereum blockchain. Data reveals that certain transactions, such as those involving Tornado Cash [91], experience significant delays. The study [90] posits an "impossibility result," stating that achieving censorship-resistance becomes unattainable if more than 50% of the validator nodes engage in direct censorship. This is not merely a theoretical concern; MEVWatch [92] has reported instances where a majority of validators engage in censorship.

The manipulative behaviors associated with MEV and censorship highlight a new challenge: the potential for a faulty majority. In traditional Byzantine fault-tolerant systems, the assumption is that up to one-third of the network's actors may be malicious, but the remaining two-thirds will act honestly [10]. MEV, however, introduces economic incentives that can lead a majority of miners to act in ways that prioritize personal gain over network health. This shift challenges these foundational trust assumptions and requires a reevaluation of how trust is established and accountability is achieved in decentralized systems.

1.3.3 Attack Resilience

Attack resilience in the distributed systems refers to the ability of a network, protocol, or application to withstand, detect, and effectively respond to malicious attempts aimed at disrupting its normal operation, compromising its integrity or subverting its intended functionality. While blockchain often takes the spotlight in discussions about Web3, it is important to recognize that Web3 encompasses a broader ecosystem, as detailed in Section 1.2.3. Resilience in Web3 involves a combination of robust architectural design, effective consensus mechanisms, rigorous smart contract auditing, and adaptive security measures that can evolve in response to emerging threats.

According to a 2021 report, fraudulent activities and various hacks led to a staggering loss of 12 billion dollars across the Web3 landscape [93]. These attacks range from exploiting simple misconfigurations to more complex ones. For instance, the DAO's reentry attack in 2016 [94] led to a loss of 3.6 million ETH tokens and necessitated a hard fork in Ethereum. Similarly, Bancor's overflow attack [95] in 2018 resulted in a loss of 1.2 million USD, underscoring the critical importance of rigorous code review and security practices.

The Web3 ecosystem is susceptible to a myriad of attacks, including but not limited to 51% attacks, phishing, smart contract vulnerabilities, and Sybil attacks. Each attack type exploits different aspects of the decentralized and open nature of Web3 technologies. Among the various types of attacks, the Sybil attack [96] stands out as one of the most fundamental exploitations. This is due to the inherent reliance on pseudonymity and the absence of centralized identity verification within decentralized systems. It is easy for a single adversary to create and control multiple pseudonymous identities, thereby gaining disproportionate influence within the network. Moreover, the versatility of Sybil attacks allows them to be adapted to target numerous aspects of Web3, from peer-to-peer networks and blockchain consensus mechanisms to DApps. This adaptability to different contexts makes Sybil attacks a critical focus for security research, as they pose a significant threat to the integrity and reliability of decentralized systems.

Various countermeasures have been proposed [97] to tackle Sybil attacks. Traditional defense mechanisms used in centralized systems, such as CAPTCHAs or identity verification processes, are not directly applicable to decentralized systems or may contradict the principles of decentralization. CAPTCHAs, for instance, depend on a centralized server to validate responses, while traditional identity verification often involves centralized databases.

Decentralized solutions include staking mechanisms, cryptographic puzzles, and decentralized identity proofs. Staking mechanisms require participants to lock up tokens as a form of collateral, making it economically unfeasible for an attacker to create numerous identities. Cryptographic puzzles require significant computational effort to solve, thereby limiting the ability of attackers to generate multiple fake identities. Decentralized identity proofs aim to verify the uniqueness of participants in a decentralized manner, ensuring that each identity is genuine without relying on a central authority. Research into reputation systems and novel consensus algorithms also holds promise. Reputation systems could provide a way to build trust within the network based on users' historical behavior, making it harder for malicious actors to gain influence.

Despite these initial proposals, we still lack a systemic understanding of the trade-offs involved in these solutions. For instance, staking mechanisms can centralize power among

wealthy participants who can afford to lock up large amounts of tokens, potentially undermining the decentralization principle. Cryptographic puzzles, while effective at limiting Sybil attacks, can be resource-intensive and environmentally unsustainable. Decentralized identity proofs and reputation, though promising, are still in their nascent stages and require more robust frameworks to ensure security without central oversight.

1.3.4 Incentive Compatibility

Incentive compatibility [98] is a fundamental concept in the design and analysis of economic mechanisms and systems. It refers to the alignment of individual participants' incentives with the overall objectives or desired outcomes of the system. In an incentive-compatible system, participants are motivated to act in ways that not only benefit their own interests but also contribute positively to the collective goals of the network or community.

In Web3 systems, incentive compatibility is achieved when rewards and penalties are structured to discourage malicious behaviour and to encourage contributions to the network's public good (e.g. transaction validation or block production). Tokens usually provide the monetary layer for these incentives. However, many token models are naïve in design and execution, concentrating on headline economics such as issuance schedules or airdrop distribution while overlooking governance, utility, and external dependencies [99]. Unsurprisingly, tokens launched without a clear utility frequently suffer from price volatility and weak adoption.

Because these models also lack robust risk-management tooling [100], they are exposed to attack patterns such as (i) the *"Token Raid Attack"*, in which a large holder deliberately depresses price to accumulate more stake or extract excess dividends; (ii) the *"Insider Attack"*, where privileged knowledge (e.g. about upcoming product releases) is used to front-run the market; and (iii) the *"Blocking Attack"*, in which a competitor amasses enough voting power to stall or veto proposals, undermining governance and eroding token value.

The intuitive defence—*"it would be too expensive for an attacker"*—is often misplaced. Real-world incidents reveal that the *effective* cost can be far lower than the nominal market capitalisation:

- **Governance capture (Token Raid).** In 2020 a coordinated acquisition of *Steem* stake, aided by exchange-held tokens, enabled a hostile takeover at a fraction of the chain's market cap [101–103].
- **Flash-loan governance attack (Blocking).** The 2022 *Beanstalk* exploit used flash loans to borrow $80 M of voting power for a single block, pass a malicious proposal, and siphon $181 M—incurring only loan fees and gas [104–106].
- **Insider manipulation.** In the first U.S. crypto insider-trading conviction, a Coinbase employee generated $1.5 M in illicit gains by leaking listing information; the limiting factor was legal risk, not capital [107, 108].

Addressing these challenges requires another approach to token design and governance. Token models must incorporate mechanisms that balance economic incentives with governance structures that mitigate risks. In that community engagement is also essential; involving the community in governance decisions ensures that the token model aligns

with user needs and expectations. While monetary incentives, such as cryptocurrency rewards, play a crucial role in motivating participants to contribute resources and uphold the system, they are not the only driving forces. In a decentralized environment, mutual cooperation is vital for achieving common objectives like block and transaction sharing.

In turn, this cooperative ethos faces significant challenges, notably the "tragedy of the commons"[109]. This phenomenon occurs when individuals, acting in their self-interest, deplete shared resources to the detriment of the collective good. This issue is not new and has been observed in various systems, including BitTorrent[15] networks where users download files but fail to seed them back into the network. In the context of Web3, similar challenges can arise, such as when participants use network resources without contributing back. This behavior can undermine the sustainability of decentralized networks.

1.4 Research Questions

This thesis seeks to address the research questions related to the foundation of Web3. These questions investigate the four current Web3 shortcomings identified in Section 1.3, which are: reliability (RQ1), accountability (RQ2), Sybil attack resilience (RQ3), and incentive compatibility (RQ4). The overarching research question guiding this thesis is: How can we establish a trustworthy foundation for Web3 by addressing its key challenges in reliability, accountability, resilience, and incentive compatibility?

[RQ1] How do blockchain systems perform in real-world scenarios in terms of throughput, latency, and scalability? In recent years, a plethora of blockchain systems and architectural designs have been introduced. While these systems often report promising metrics such as high transactions per second, low latency, and superior scalability, these figures are frequently derived from idealized or controlled environments. The true test of a blockchain's capability lies in its performance under real-world conditions, which can be significantly influenced by factors such as network congestion, varying node distribution, and diverse transaction patterns. Currently, the field lacks a universally accepted benchmarking framework that allows for the equitable comparison of blockchain systems. This research question seeks to address this gap by conducting an experimental analysis of the most popular blockchain implementations.

[RQ2] What mechanisms can be designed to achieve accountability of blockchain validators? The integrity of blockchain systems heavily relies on the behavior of validators, whose roles are critical in transaction validation and block creation. However, the decentralized nature of these systems often complicates the enforcement of accountability. Particularly, there is no reliable way to detect and account for Miner Extractable Value (MEV) and transaction censorship. Classical accountability mechanisms, such as PeerReview [87], which may work well in centralized systems, are not directly transferable to the blockchain context due to its inherent decentralization and the need for trustless interactions. This research question probes into the design of novel mechanisms that can ensure the actions of the validators nodes are transparent and that they can mutually hold themselves accountable.

[RQ3] How can reputation systems be effectively integrated into Web3 ecosystems to enhance resilience against Sybil attacks? Sybil attacks present a formidable challenge in decentralized Web3 environments, exploiting the ease of creating numerous pseudonymous identities. Integrating reputation systems offers a promising countermea-

1

sure by assigning trust scores to entities based on their network behavior and contributions. However, these systems in turn face the challenge of reputation manipulation, where malicious entities artificially inflate their trust scores through Sybil or collusive activities. This research question delves into the design and implementation of robust reputation-based systems within Web3, specifically tailored to withstand both Sybil and reputation manipulation attacks. The focus is on developing mechanisms that accurately assess participant behavior, dynamically adjust reputation scores, and incorporate safeguards against manipulation, thereby ensuring the reliability and integrity of the reputation system as an effective defense against Sybil attacks.

[RQ4] What incentive mechanisms can be designed to sustain cooperation within decentralized systems? Cooperation is essential in decentralized systems, driving collective contributions to network functionality and resilience. However, aligning individual incentives with collective goals to sustain cooperation is a complex challenge. This research question explores the development of technically efficient incentive mechanisms that encourage cooperative behavior in decentralized ecosystems. The aim is to craft incentive structures that are lightweight and introduce minimal overhead, to ensure system efficiency. These mechanisms should also possess the capability to progressively detect and isolate nodes that engage in 'freeriding' behavior, thereby safeguarding the network against exploitation and ensuring that participants who contribute positively are duly rewarded.

1.5 Research Methodology

This thesis employs research and engineering methods typical to the fields of networking and distributed systems, focusing on experimental research methodology and qualitative analysis of system components. The research questions are answered through a process of mechanism design, implementation, and evaluation. A common approach for these chapter is to demonstrate a practical solution, and then analyze (parts of) the entire design space. To ensure reproducibility, we release both the prototype code and the datasets used, which are available through 4TU.ResearchData.

Each prototype to aid answer the research questions is developed in Python. Our experiments are based on data traces from real users. Prototypes are built using networking and overlay primitives developed and maintained by our lab [113]. Evaluations are conducted using a nationwide compute cluster DAS5 [114], and experiments are set up and run using the Gumby framework, also developed and maintained by our lab [115].

Table 1.2 shows the research methods and availability of each artifact for each reserch question. The research led to four artifacts, each developed to answer a specific research

Table 1.2: Mapping of research questions to chapters, evaluation methods, system names, and artifact availability.

RQ	Chapter	Method	System Name	Availability
RQ_1	2	emulation	Gromit	[110]
RQ_2	3	emulation	LØ	[111]
RQ_3	4	simulation	MeritRank	[112]
RQ_4	5	emulation + deployment	Coop	[111] & [113]

question, using three different evaluation methods:

Simulation: For RQ_3, we use a discrete-event simulator to mimic the behavior of a real system based on traces from real-world interactions on MakerDAO forums.

Emulation: For RQ_1, RQ_2, and RQ_4, we use emulation on DAS5. Emulation involves running full system prototypes in a controlled environment using real-world latency traces from the WonderNetwork [116] dataset. $RQ1$ deals with existing implementations of blockchain systems. Datasets from Bitcoin and Ethereum transactions are used to mimic real-world workloads. The specific details on which blocks are used are specified in the technical chapters.

Deployment: For RQ_4, we deploy our mechanism to Tribler client [117]. This is done by creating a experimental release of the client and analyzing later the usage and results. We answer the research question by analyzing the the traces for over a year run of the client.

1.6 Thesis Outline and Contributions

This section summarizes the chapters in this thesis. Table 1.2 outlines the connections between the research questions and the chapters. Chapter 2 investigates the *reliability* of blockchain systems. Chapter 3 addresses the *accountability* of block creators. Chapter 4 demonstrates *Sybil attack resilience* in the context of DAOs. Finally, Chapter 5 develops an *incentive-compatible* transaction dissemination protocol for permissionless blockchains.

Chapter 2: With the rapid growth in the number of blockchain protocols, driven by interests in cryptocurrencies, decentralized finance, and identity systems, there is a significant gap in research that systematically compares these solutions. To bridge this gap, we designed Gromit, a generic framework for analyzing blockchain systems. Gromit views each blockchain system as a 'transaction fabric', evaluating the performance at the system level. We conducted a benchmark study of selected state-of-the-art blockchain designs. Our experimental analysis reveals that performance for most evaluated blockchain systems degrades as the number of peers increases. We provide insights into real-life bottlenecks in terms of throughput and latency of the underlying consensus mechanisms.

This chapter is based on the publication:

Bulat Nasrulin, Martijn De Vos, Georgy Ishmaev, Johan Pouwelse, "Gromit: Benchmarking the Performance and Scalability of Blockchain Systems," *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2022.

Chapter 3: Miner Extractable Value (MEV) manipulations have generated over 320 million USD in revenue for bots and miners, with over 90% of blocks produced on Ethereum containing MEV-related transactions. These manipulations can lead to system congestion, inflated transaction fees, and overall instability. We postulate that the lack of accountability in the initial transaction handling phase (pre-consensus phase) of permissionless blockchain protocols is the main enabler for MEV. The consensus phase is responsible for validating and ordering blocks of transactions. The fairness of individual transactions is not directly

1

addressed at this phase. In contrast, at the standard pre-consensus phase nodes implicitly trust each other to correctly relay transactions. This trust can be exploited, allowing block creators to censor, reorder, or inject transactions arbitrarily. We develop LØ pre-consensus protocol to ensure accountability and detect transaction manipulations. LØ operates by compelling miner nodes to log all received transactions into a secure mempool data structure. We demonstrate that LØ is both bandwidth and memory efficient. Our system can expose all malicious nodes, even in a network under adversarial conditions.

This chapter is based on the publication:

Bulat Nasrulin, Georgy Ishmaev, Jérémie Decouchant, and Johan Pouwelse. "LØ: An Accountable Mempool for MEV Resistance." *ACM Middleware*, 2023.

Chapter 4: Reputation schemes are considered promising solutions to address the limitations of tokenomics. However, when implemented naively, they introduce significant challenges, particularly concerning Sybil attacks. To tackle this, we designed MeritRank, a novel Sybil-tolerant feedback aggregation mechanism for reputation systems. Instead of attempting to prevent Sybil attacks outright, MeritRank aims to limit the benefits attackers can gain from such attacks. We simulated user interactions using data from MakerDAO and found that MeritRank effectively reduces the benefits an attacker can gain from a Sybil attack to the amount they genuinely deserve based on their actual contributions.

This chapter is based on the publication:

Bulat Nasrulin, Georgy Ishmaev, Johan Pouwelse, "MeritRank: Sybil Tolerant Reputation for Merit-based Tokenomics," *Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2022.

Chapter 5: The inherent openness and decentralized nature of peer-to-peer networks present a significant challenge: ensuring sustained cooperation among peers. Traditional P2P systems have largely depended on principles of altruism and reciprocity. While these principles foster a sense of community, they are often insufficient in preventing selfish behaviors, where nodes prioritize their interests over the collective good, leading to potential degradation in network performance and fairness. We develop Coop, a novel protocol designed to reward sustained cooperation among peers, even in the face of selfish peers who might withhold resources for personal gains. This protocol operates by aggregating local evaluations of cooperation, effectively mitigating vulnerabilities associated with Sybil and misreporting attacks. The solution hinges on a protocol that evaluates and ranks the relative contributions of overlay neighbors within the P2P network. The network dynamically adjusts the P2P overlay to account for and counteract the presence of selfish peers. We demonstrate the effectiveness of Coop by closely emulating the Bitcoin network and showing how the quality of service for selfish nodes is eventually degraded.

This chapter is based on the publication:

Bulat Nasrulin, Rowdy Chotkan, Johan Pouwelse, "Sustainable Cooperation in Peer-To-Peer Networks," *IEEE Conference on Local Computer Networks (LCN)*, 2023.

2

Gromit: Benchmarking the Performance and Scalability of Blockchain

The growing number of implementations of blockchain systems stands in stark contrast with still limited research on a systematic comparison of performance characteristics of these solutions. Such research is crucial for evaluating fundamental trade-offs introduced by novel consensus protocols and their implementations. These performance limitations are commonly analyzed with ad-hoc benchmarking frameworks focused on the consensus algorithm of blockchain systems. However, comparative evaluations of design choices require macro-benchmarks for uniform and comprehensive performance evaluations of blockchains at the system level rather than performance metrics of isolated components. To address this research gap, we implement Gromit, a generic framework for analyzing blockchain systems. Gromit treats each system under test as a transaction fabric where clients issue transactions to validators.

We use Gromit to conduct the largest blockchain study to date, involving seven representative systems with varying consensus models. We determine the peak performance of these systems with a synthetic workload in terms of transaction throughput and scalability and show that transaction throughput does not scale with the number of validators. We explore how robust the subjected systems are against network delays and reveal that the performance of permissoned blockchain is highly sensitive to network conditions.

2.1 Introduction

The rapid growth in the number of blockchain protocols in the past few years has been boosted by the interest in crypto-currencies, decentralized finance, and identity systems. The solutions are mostly empirically driven, with direct economic incentives stimulating engineering experiments. To date, there are more than 700 different blockchain and distributed ledger platforms offering native digital assets and products.Many of these solutions deploy original families of consensus protocols or significant modifications of popular protocols, with variations in scalability, performance, and decentralization guarantees. These developments outpace systematization and research on inherent trade-offs of different design choices [118].

The absence of benchmarking solutions for comprehensive comparative analysis of various protocols is a particularly problematic omission, given the cumulative marketcap of 1,468 trillion $ for these projects. More fundamentally, this systematization gap hampers our ability to tackle the increasing complexity of blockchain systems and make conscious design choices in blockchain engineering. Developers of these protocols often provide performance metrics of blockchain solutions as declarative whitepapers that do not pass the standards of peer review and reproducibility, calling into question the reliability and objectivity of these measurements. For instance, it is a common practice to provide performance metrics of an isolated component and report them as a system-wide performance metric [119–121]. This practice often leads to false impressions of the end-to-end system performance.

Few available benchmarking solutions, such as Blockbench [122] and Hyperledger Caliper [123] focus on narrow sets of permissioned consensus protocols or DAG-based protocols as DAGBENCH [124]. There is also a noticeable deficit of academic research in macro benchmarks for blockchain systems. Existing studies are rather limited in scope either focusing on specific protocols such as Hyperledger [125, 126] or Ripple [127], or reusing Hyperldeger Caliper [128] and Blockbench [122]. BCTMark is one of the few comparative benchmark studies which compares three different protocols, including permissionless Ethereum blockchain [129]. The authors in this study highlight the necessity to extend the comparison set and include more metrics such as partition tolerance. The most recent systematic survey on performance evaluation of blockchain systems demonstrates that available comparative studies are rather limited in scope both in terms of compared systems and depth of analysis, focusing on isolated layers of blockchain systems [118]. To address this research gap, we design a benchmark that is comprehensive in scope, allowing us to stress-test the system under different network conditions.

We introduce Gromit, a generic benchmarking framework that allows a performance evaluation of *any* blockchain solution. Gromit treats each system under test as a transaction fabric, which means a transaction processing system where a group of peers continuously reach a consensus on transactions submitted by clients. Gromit analyses performance metrics related to transactional data, specifically throughput and latency. As we will show, this data alone can reveal the limitations of various aspects of the system.

We show the applicability of Gromit and conduct the *largest blockchain benchmarking study to date*. Our benchmark involves seven major blockchain solutions with different consensus algorithms. We determine the peak performance of these blockchain systems for different numbers of peers and without any modifications to the source code. A key finding is that the performance for most evaluated blockchain systems *degrades* when the

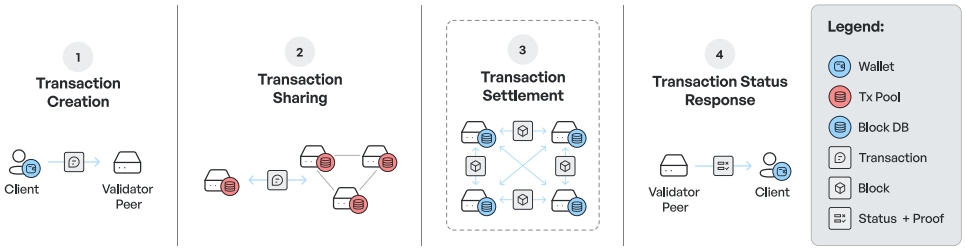


Figure 2.1: Our abstraction model of blockchain system as a transaction fabric, comprising four stages. The dotted line highlights the focus of most whitepaper benchmarks.

number of peers increases. We also reveal the effect of network delays and an increase in system load on the distribution of end-to-end transaction latencies, yielding valuable insights into the real-life bottlenecks of underlying consensus mechanisms. While most performance bottleneck research focuses on the consensus layer, a system-wide stress test can reveal bottlenecks in other layers, e.g., in the persistence layer.

The contribution of this work is two-fold:

1. We design and implement Gromit, a benchmarking framework that enables an analysis of *any* blockchain system (Section 2.3).
2. We conduct *the largest blockchain benchmark to date*, measuring the performance of seven prominent blockchain systems (Section 2.5).

2.2 Blockchain as a Transaction Fabric

In this work, we view a blockchain system as a *transaction fabric*. Figure 2.1 visualizes this abstraction model, which illustrates a typical transaction life cycle.

In our model, we distinguish between clients and peers. Clients are instances that create transactions and send them to peers. An example of a client is a light wallet or lightweight nodes in a Bitcoin network. Peers in our model are responsible for processing and validating transactions in a shared, decentralized network. Thus, we call them *validator peers*. In some blockchain systems, they are also referred to as miners.

2.2.1 Transaction Life Cycle Model

Transaction Creation

A *transaction* contains logic that modifies the system state, e.g., by transferring an asset to another account. Each transaction is cryptographically signed with the private key of the issuing client to ensure authenticity. Blockchain solutions usually provide *Wallet* API's for clients to submit their transactions, e.g., with an RPC endpoint or REST endpoint.

Transaction Sharing

Blockchain systems employ complex transaction sharing mechanisms. Permissionless blockchains typically use a global gossip protocol to share transactions over a structured or unstructured overlay. Permissoned blockchains are deployed in a more controlled network environment and, as a result, might share transactions using a broadcast algorithm.

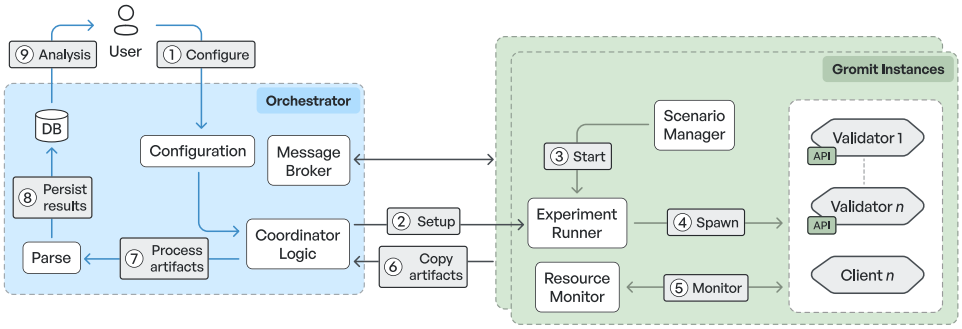


Figure 2.2: Architectural overview and process flow of Gromit, our benchmarking framework for blockchain systems.

The transactions are stored in a datastore, often called a transaction pool (*TxPool*). The transaction pool is a temporary store used to queue or preprocess transactions before the network validates them.

Transaction Settlement

A consensus algorithm is a crucial part of the blockchain system as a mechanism for achieving security and liveness[130]. The intermediate outcome of the consensus process in blockchain systems is a set of valid transactions. Valid transactions are stored in a tamper-proof distributed ledger, a replicated data structure maintained by validators. The system discards invalid transactions.

Blockchain solutions typically bundle valid transactions in *blocks*, interlinked in a hash chain, and stored in a local database (*Block DB*). Each block in a hash chain contains the cryptographic hash of the previous block, making illegitimate modifications of the chain detectable. Some blockchain-like systems adopt a different organization of the distributed ledger, e.g., by maintaining a Directed Acyclic Graph (DAG) [124].

Transaction Status Response

After the transaction is settled the client waits for a transaction approval (or rejection), received from validator peers. Some blockchains also include a *proof* in the response that proves that a transaction is finalized.

2.2.2 Transaction Performance Indicators

Our approach is to analyze the performance of current blockchain solutions through *transaction benchmarking*. The speed at which a blockchain system processes transactions is a defining metric for blockchains. High transaction latencies directly impact end-users experience, e.g., the relatively high finalization times of Bitcoin transactions (10 minutes) make it unsuitable for interactive trade [131]. We include all stages of the transaction life cycle in our measurements, rather than focusing on the performance of the consensus layer only.

We obtain insights into the limitation of blockchain systems by measuring peak performance and associated transaction latencies. We also measure performance under different

network conditions, such as changes in the geographical distribution of the overlay network. Our experiments in Section 2.5 highlight that these metrics can reveal performance bottlenecks and act as a guideline for optimization efforts.

2.3 Gromit: A Generic Framework for Blockchain Benchmarking

We design and implement Gromit, the first generic framework for blockchain benchmarking. Gromit is designed for quick experiment iterations, utilizing a domain-specific language to devise experiments. Our framework, implemented in Python, is based on the abstraction model discussed in Section 2.2.

Gromit provides developers and researchers with the necessary tools to design benchmarks targeting blockchain performance. Gromit spawns client and validator processes on remote servers and coordinates interactions between running processes, e.g., transaction issuing.

2.3.1 Experiment Flow

Figure 2.2 shows the architecture of the framework and the flow of an experiment. This diagram is described next.

Experiment Setup. Before an experiment starts, it spawns a dedicated process, called the *orchestrator*, that setups the environment on remote servers and handles cross-server communication. The orchestrator reads the configuration file associated with the experiment. This file specifies the network addresses of remote servers and the number of clients and validators that should be started (step ①). The orchestrator then copies the source code and all necessary experiment files to the remote servers using `rsync`. Next, the coordinator setups the environment on the specified servers over an SSH connection (step ②). This step includes the installation of required system packages and the generation of a genesis file. This file describes the initial state of the blockchain system and can pre-load specific accounts with assets. The orchestrator then starts a *instance* for each client or validator on the remote server and assigns an identifier to each running instance. The scenario manager, part of the logic of a Gromit instance, parses a provided scenario file and starts the experiment (step ③).

Scenario Files. A user describes the actions performed during an experiment with a *scenario file*. The scenario manager parses this file and schedules actions using the `asyncio` library.

```
1 @8   init_blockchain_config {1-10}
2 @12  start_validator {1-10}
3 @20  start_client {11-20}
4 @80  start_creating_transactions {11-20}
5 @100 stop
```

Example 2.1: An example of a scenario file in Gromit, describing an experiment with 10 clients and validators.

An example scenario file associated with a simple blockchain benchmark is given in listing 2.1. Actions and their timestamps are explicitly denoted. A user can schedule an

action to execute on a subset of all validators. Scenario files can be machine-generated and are a flexible approach to devising and conducting experiments.

Experiment Runner. The experiment runner spawns validators and clients as a subprocess during each experiment run (step ④). Clients interact with validators through the exposed API endpoint. We simulate a client in Gromit as a procedure that issues transactions. During an experiment, Gromit instances can share data over a TCP connection through a message broker. We use the message broker functionality to share the credentials of pre-defined blockchain accounts with clients. Gromit contains tools to gracefully terminate a running experiment, e.g., when a particular condition is not met.

Gromit also provides utilities to track system resource usage. Gromit instances monitor CPU, memory, disk, and network usage using the `procfs` library (step ⑤). These metrics allow us to estimate the system resource usage of blockchain systems. Developers can easily extend Gromit to monitor specific metrics, for example, the number of inbound network messages for a particular blockchain system.

Collecting Experiment Results. When the experiment ends, the orchestrator copies all generated artifacts from the remote nodes using `rsync` (step ⑥). These artifacts include the data generated by the blockchain systems and the data output by the Gromit instances, e.g., monitoring statistics. This data is parsed by the orchestrator (step ⑦) and generates human-readable graphs. Finally, the data is stored in a database (step ⑧) and is ready for analysis by the user (step ⑨).

2.3.2 Integrating Blockchain Systems into Gromit

To show practicality of Gromit, we have integrated seven prominent blockchain systems into it. The integration requires no change in the source code of the blockchain systems. This allows to benchmark the blockchain system as close to the deployed systems as possible.

The design of Gromit is modular, and developers can implement *modules* that enrich an experiment with more functionality, for example, bandwidth monitoring. Integration of a blockchain system requires a developer to subclass the `BlockchainModule` and to implement the `init_configuration`, `start_validator`, `stop_validator`, and `parse_ledger` methods. Gromit enables developers to specify custom network topologies to connect validators among each other. We refer readers who wish to integrate a particular blockchain system to the Gromit documentation.

Besides supporting blockchain systems, Gromit has support for generic experiments with distributed systems. For example, we have used Gromit to conduct experiments with peer-to-peer protocols on custom infrastructure.

2.3.3 Transaction Analysis

A workload during a Gromit benchmark consists of transactions issued by one or more clients. All transactions can be submitted to one validator peer or can be evenly spread among the peers. In line with related work, we mainly gauge blockchain performance using two metrics, transaction throughput, and transaction latency. Specifically, we consider the peak transaction *throughput*, which is the maximum rate at which the system can process transactions before getting congested. Second, we analyze the *latency* of transactions,

which is the time between submitting a transaction and its irreversible inclusion in the distributed ledger.

We store the timestamp at which a client has submitted the transaction to a validator to determine transaction latency. However, determining the finalization time of transactions can be complicated since some blockchain systems do not expose granular, temporal information. We use the following two approaches to determine the latency of individual transactions. Our first approach is to inspect the resulting distributed ledger after all transactions have been issued (this logic should be part of the `parse_ledger` method). We then compute transaction latency based on timestamps included in ledger data elements (e.g., blocks). However, this approach is useful only if the blockchain system annotates data elements in the distributed ledger with a timestamp. The second approach suggests clients periodically poll the blockchain system to determine if a transaction has been confirmed.

2.4 Blockchain Consensus Model

Since the introduction of Bitcoin in 2008, there have been many proposals for new blockchains and consensus mechanisms. Some proposals have materialized into operational systems, whereas other ones are only theoretically analyzed. The proliferation of blockchain solutions makes it infeasible to conduct a benchmarking study with all available systems.

Based on taxonomy of different consensus mechanisms used in blockchain systems [130] we consider seven prominent consensus models and select a representative blockchain system for each consensus model. Our selection process is based on the economic magnitude, adoption, maturity of the system, and the protocol's academic significance. Table 2.1 lists the representative system for each considered consensus model. For each system, we also show the evaluated version of the software, the principle underpinning each consensus model, how the group of validators is formed, and the year in which the first software commit has been made (as an indicator of matureness). We are aware of state-of-the-art blockchains that use techniques such as layer-one scaling to improve throughput, e.g., OmniLedger [137] and RapidChain [138], or layer-two scaling solutions [139]. However, the primary focus of our benchmarking study is on deployed layer-one blockchain systems. We now elaborate on each consensus model and refer the reader to [130] for a more extensive overview of blockchain consensus models.

Proof-of-Work (PoW) is the oldest consensus mechanisms designed explicitly for blockchain systems. PoW is used in blockchains such as Bitcoin [1] and Ethereum [2] and to date remains a standard approach to build open blockchain systems. PoW scales well in the number of participants: the Bitcoin network has over 11'000 operational

Table 2.1: The seven selected blockchain systems and consensus models analysed in this work.

Blockchain System	Consensus Model	Consensus Principle view	Validator Group Foundation Formation	First commit (year)
Ethereum (v1.9.24) [2]	Proof of Work (PoW)	Resource-based lottery	Resource mining	2013
Algorand (v2.3.0) [132]	Proof of Stake (PoS)	Random selection of leaders	Stake-based enrolment	2019
BitShares (v5.0.0) [133]	Delegated Proof of Stake (dPoS)	Rotating leader	Election by stakeholders	2015
Diem (v1.1.0) [134]	PBFT, based on HotStuff [135]	Leader-based	Enrolment by an authority	2019
Stellar (v15.1.0) [120]	Federated Byzantine Agreement (FBA)	Quorum Intersection	User-defined quorums	2014
Hyperledger Fabric (v1.4.9) [46]	CFT, based on Raft [136]	Leader-based	Enrolment by an authority	2016
Avalanche (v1.1.1) [119]	Meta-Stable Consensus (MSC)	Network subsampling	Stake-based enrolment	2020

miners. However, PoW is known to consume large amounts of energy, and the achievable transaction throughput is theoretically limited to around 60 transactions per second [42].

Proof-of-Stake (PoS), initially proposed by the Peercoin cryptocurrency [54], is an alternative consensus mechanism that addresses the excessive resource usage by PoW. PoS is typically based on a random, periodic selection of a leader. This selection process is weighted by the participants' stake in the system, e.g., by the number of assets owned, or by the age of possessed assets. Algorand is one of the most prominent PoS-based blockchains and leverages Verifiable Random Functions (VRFs) for the leader election process [132].

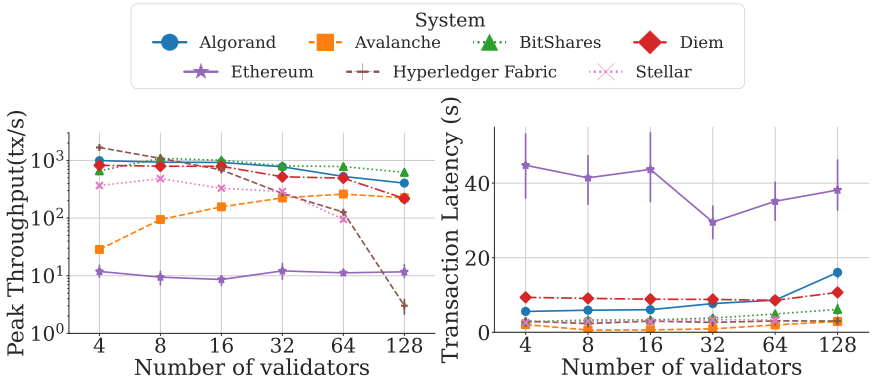
Delegated Proof-of-Stake (dPoS) is a consensus mechanism where stakeholders elect a group of *delegates* through voting. Voting decision can, for example, be based on community engagement. Since the set of validators participating in consensus remains relatively small compared to PoS-based solutions, dPoS-based consensus has in theory a better potential to scale. BitShares is one of the most mature blockchains using dPoS consensus [133].

Practical Byzantine Fault Tolerance (PBFT) is a consensus algorithm introduced in 1999 [10]. PBFT is specifically designed for networks with static and pre-approved membership and has been revised for adoption in blockchain environments. Recent advancements have resulted in HotStuff [135], a consensus protocol based on PBFT that reduces communication overhead and increases throughput. HotStuff is at the core of Diem, a permissioned distributed ledger maintained by a consortium led by Facebook [134].

Federated BFT (FBFT) is a consensus model that distinguishes itself from the approaches mentioned earlier by having validators explicitly specifying trust relations. Stellar is one of the first systems to adopt FBFT consensus [120]. The Stellar Consensus Protocol (SCP) leverages a federated voting approach in which each validator votes on statements while ensuring that no two members of an overlapping quorum can confirm contradicting statements.

Crash-tolerant Consensus (CFT) is a consensus approach widely used to achieve fault tolerance, e.g., by Apache Kafka [140]. Unlike PBFT, CFT is not resistant against arbitrary (Byzantine) behaviour but can withstand crash-stop failures of participants. Notable algorithms achieving CFT are Paxos [141] and Raft [136]. Hyperledger Fabric, one of the most prominent industrial blockchains, is currently using Raft [46].

Metastable Consensus (MSC) is a family of consensus algorithms that leverage network subsampling techniques to determine the validity of a transaction. The idea is to repeatedly sample random validators in the network and to steer correct nodes to a common decision. Avalanche is one of the most mature blockchain solutions to leverage MSC consensus and maintains a DAG data structure to store transactions [119].



(a) Peak transaction throughput

(b) Transaction latency

Figure 2.3: The peak throughput and transaction latencies of evaluated blockchain systems with the number of validators.

2.5 Blockchain Performance Evaluation

We conduct a diverse set of experiments with Gromit to reveal the performance characteristics and limitations of seven blockchain solutions. To the best of our knowledge, We are the first to perform a large-scale performance benchmark of blockchain solutions.

Our experiments answer the following questions: How does the increase in the number of validators affect the system performance? What is the peak performance of each system under a heavy system load? What is the impact of a network delay on systems' performance? How consistent are our performance results compared to previously reported values?

Throughout the section we use two variables for our experiments: n indicates the number of validators and λ signifies the transaction throughput.

2.5.1 Setup and Transaction Workload

All experiments are conducted on four HPE DL385 Gen10 servers, located within the same data centre and inter-connected with 10GB Ethernet links. Each server is equipped with 128 AMD EPYC 7452 CPUs, has 512GB of DDR4 memory, and runs Debian 10. During our experiments, we deploy each blockchain system with its source code unmodified. For each system, we use the default settings provided by the systems. Each experiment starts with only the genesis block included in the blockchain. We use a random network topology where each validator is connected to 10 other random validators.

Assumptions and Scope. We deliberately benchmark under *homogeneous* validator and client configurations because this provides a clean, controlled baseline from which differences in protocol design—not hardware heterogeneity—drive the measured effects. In practice, heterogeneity typically *reduces* throughput and *inflates* latency, so the numbers we report should be regarded as an upper bound; real-world deployments with slower or unevenly provisioned nodes will only accentuate the bottlenecks we expose. Likewise, we

treat the end-to-end transaction path as a monolithic pipeline and do not attempt to allocate latency to each fabric phase unless the data clearly diverge from published results. This pragmatic choice keeps the experimental matrix tractable and, more importantly, avoids over-fitting explanations to noise while still allowing us to zoom in on specific phases whenever discrepancies with the literature appear (e.g., the API bottleneck in Avalanche).

We use Gromit to subject each system to a synthetic transaction workload, issued by up to 64 clients. To ensure an equal load on each validator, a client submits transactions to the validator with ID $i \equiv c \pmod{n}$ where c is the ID of the client and n the total number of validators. Each transaction is submitted to exactly one validator. Transactions are submitted during a two-minute period, after which we wait an additional minute for all transaction to be finalized.

We use simple asset transfers as a performance baseline. In our workload, a transaction issued by a client involves an asset transfer of a small, fixed amount to another account; the client counterparty is fixed throughout the experiment. We ensure that each client has sufficient funds to spend during the experiment. For Ethereum and Hyperledger Fabric, transactions involve the transfer of an ERC20 token.

2.5.2 Determining Peak Transaction Throughput

To determine peak transaction throughput, we gradually increase the system transaction rate in steps of 100 transactions per second (tx/s). Based on the reported statistics by Gromit, we estimate the peak transaction load that each system is still able to process during a sustained period. If the system has any unconfirmed transactions after our two-minute period, we consider the system as "saturated". We evaluate the peak throughput of each system with an increasing number of validators (n). We provide each system with an equal amount of resources and ensure that the resource usage of evaluated systems (CPU power, disk space, and memory) does not exceed the available resources. Due to excessive resource usage of the Stellar software, we are only able to run Stellar with up to 64 validators. We run each experiment at least five times and average all results. Appropriate graphs are annotated with 95% confidence interval markers.

Finding 1. *Adding validators does not have a significant positive effect on the achievable peak transaction throughput of the evaluated systems.*

Figure 2.3 shows the result of our scalability experiment as n grows, in terms of peak transaction throughput and transaction latency. Figure 2.3a shows the peak transaction throughput of evaluated systems (with a horizontal and vertical log-axis). We notice that none of the evaluated systems can process over 1'000 tx/s with $n = 128$. In general, the transaction throughput of most of the systems is capped between 500 and 1'500 tx/s. Except for Ethereum, the peak transaction of all blockchains is *decreasing* as n increases. Specifically, Hyperledger Fabric shows a severe degradation in performance when $n > 8$, and is just capable of processing 2 to 4 tx/s with $n = 128$. We believe that this is caused by the underlying consensus model of Hyperledger Fabric, Raft, which does not scale well with the number of validators [136]. Of all systems, Ethereum has the lowest transaction throughput (around 10-20 tx/s), yet manages to keep stable throughput with the increase

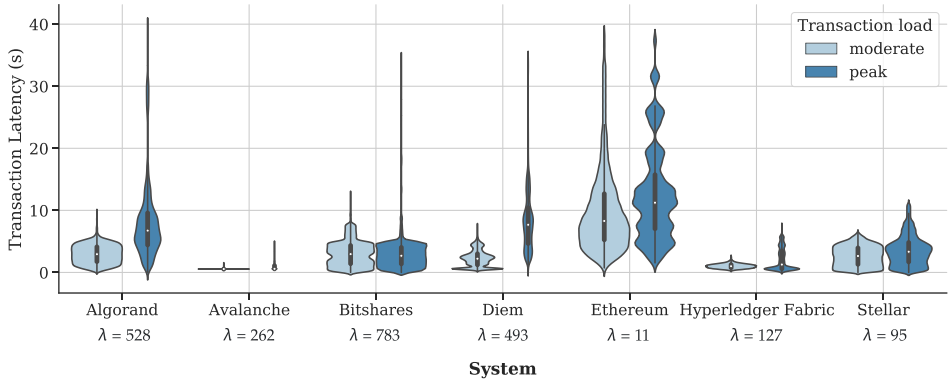


Figure 2.4: The distribution of transaction latencies for systems, under peak load (λ tx/s) and moderate load ($\frac{\lambda}{2}$ tx/s) ($n = 64$).

in the number of validators. Peak throughput of Avalanche increases up to $n = 64$, but then degrades for $n = 128$.

Finding 2. *For Avalanche, BitShares and Hyperledger Fabric, we observe significant discrepancies between the peak throughput found by us and previously reported values.*

The previous experiment estimates the peak throughput of blockchain systems, without modification to the source code and under default settings. We now compare the performance results with values reported by other literature. The performance, i.e. peak throughput, is typically determined through an evaluation by system developers themselves. For Stellar, we could not find reliable benchmarking results, making this work the first benchmarking study of Stellar. We are interested to see if there are significant inconsistencies with performance metrics reported by these studies.

Our performance results are comparable with results reported for Ethereum (4-40 tx/s [142]), Algorand (880 tx/s [132]) and Diem (200-1'000 tx/s [143]). However, we notice that previously reported values for some systems are significantly higher than our findings, specifically for Avalanche (7'000 tx/s, 26x higher [119]), BitShares (3'300 tx/s, 3x higher [144]) and Hyperledger Fabric (3'500 tx/s, 2 times higher [46]). For each system, we now explain these inconsistencies with additional experiments and analysis.

Avalanche. The relative low throughput of Avalanche surprises us and warrants further performance analysis. Since we noticed that each validator node is fully utilizing a CPU core, even with $n = 4$ and 32 tx/s, we perform a CPU analysis of deployed validators using the `pprof` profiler. We find that around 60% of CPU time is spent on hash computations using the `argon2` algorithm [145]. This CPU consumption originates from the API provided by Avalanche validators. Specifically, each validator maintains a keystore with credentials that is managed by end users; interactions with that keystore, e.g., accessing a private key, requires the user to include the password hash in the request. Consequently, many parallel requests to the API by clients cause severe performance degradations.

To analyse the impact of password hashing, we recompile Avalanche with this hash verification disabled and re-run our experiment. We do not observe a significant increase in transaction throughput. However, this reveals another performance bottleneck, originating from the verification of transactions, consuming around 90% of CPU time. Since Avalanche transactions are linked in a DAG structure, incoming transactions require the validation of parent transactions, which is a resource-intensive, recursive operation. We believe this can be addressed with further engineering efforts, e.g., queueing the verification of transactions.

BitShares. We further analyse the reported throughput of BitShares and found that the peak transaction throughput (3'300 tx/s) is not an accurate performance indicator since the sustained throughput throughout the experiment is only around 450 tx/s. Specifically, the achievable throughput of the BitShares consensus algorithm seems to be predicated by the speed of the slowest consensus participants in terms of connectivity and CPU resources. When operating BitShares in a heterogeneous environment, this can result in significant deviations in transaction throughput.

Hyperledger Fabric. We further analyse the results reported in the work of Androulaki et al. [46] and Blockbench [122]. We find that their work evaluates an early implementation of Hyperledger Fabric using a different consensus model (Zookeeper or PBFT). As such, these results are not directly comparable.

We also present the following two reasons to explain the discrepancies in reported and observed throughput numbers:

1. *Experimental Software vs Production.* Many of the throughput numbers reported by system developers are extracted using a premature or even incomplete software implementation. As such, we argue that the values found by our experiments are a more accurate reflection of the achievable throughput in a production environment. Additionally, we noticed that some solutions (Diem and Stellar) have built-in measures that artificially lower the achievable throughput, likely to ensure safety properties or to prevent attacks in a production environment. A similar insights were observed in [146].

2. *Client-Validator Interaction.* In our experiments, clients submit transactions to the API exposed by the system, whereas other studies might directly inject transactions in the validator process. API-based interaction adds additional overhead as the request needs to be processed, and this approach therefore is likely to lower the peak throughput of the system. However, this approach resembles how users interact with validators when a blockchain system is Internet-deployed.

2.5.3 Transaction Latency

Finding 3. *For all evaluated systems, the average transaction latency under peak load is largely independent of the number of validators.*

Figure 2.3b shows the average transaction latency under peak load, as n increases. Except for Ethereum, the average transaction latency of evaluated systems is around or below ten

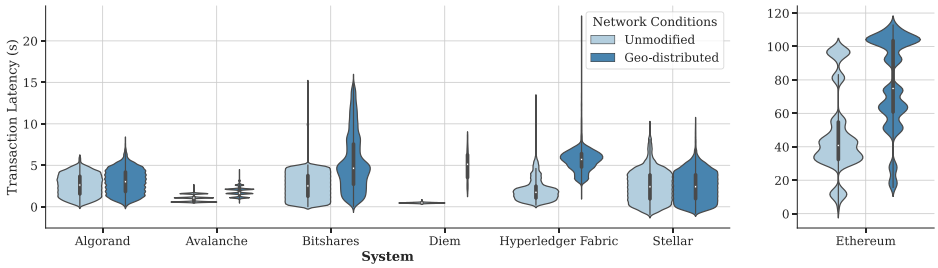


Figure 2.5: Transaction latencies without network modifications and in a geo-distributed setting ($\lambda = 64$ tx/s, $n = 32$). The transaction latencies of Ethereum are shown in the right plot.

seconds. The variance between different runs is relatively low, except for Ethereum. In general, the average transaction latency increases as the number of validators grows. The consensus algorithm underlying BitShares, Algorand and Stellar progresses in five-second rounds, theoretically resulting in an average transaction latency of 2.5 seconds. We see that the transaction latency of Algorand increases from 5 seconds for $n = 4$ to 15 seconds with $n = 128$, suggesting that consensus rounds take longer to complete. For BitShares and Stellar, this increase is less pronounced.

Finding 4. *The variance of transaction latencies for Algorand and Diem increases significantly under peak load, compared to a moderate load. However, the transaction latencies of BitShares and Ethereum are largerly indifferent towards the system load.*

We visualize the distribution of transaction latencies for each system to explore further the effects of increasing the system load on the transaction latency. We consider both peak and moderate loads, the latter being defined as half the determined peak load. Figure 2.4 shows this distribution in a violin plot. We observe that Algorand, BitShares, Hyperledger Fabric, and Stellar transaction latencies are roughly uniformly distributed under moderate load. These systems adopt a round-based consensus approach, with a target of around five seconds for Algorand, BitShares, and Stellar, and one second for Hyperledger Fabric. For these systems, most transactions are usually confirmed within the current or next consensus round relative to transaction submission. The distribution of transaction latencies transforms as the system is subjected to a peak load. Figure 2.4 shows that the finalization of Algorand transactions is being deferred to later rounds: 6% of Algorand transactions have a transaction latency above 20 seconds. This effect is less pronounced for BitShares and Stellar.

Increasing the system load impacts the latency distribution of Diem transactions. Further investigation reveals that the round duration in Diem adjusts to the system load. As more validators join the network and as more transactions are submitted to Diem validators, the round duration increases to ensure transactions can be processed on time. Nonetheless, 30% of all issued transactions in Diem are confirmed only after 10 seconds under peak load, whereas the system can handle all submitted transactions within 7 seconds

under moderate load.

2.5.4 Impact of Network Delays

Finding 5. *Adding network delays has a minimal effect on the transaction latencies of Algorand and Stellar. However, Avalanche and Diem are extremely sensitive to network delays.*

Finally, we modify the network settings using the `netem` Linux kernel module[1] and measure the impact of a network delay on the transaction latency for integrated blockchain systems. This experiment reveals how different blockchain systems are sensitive to deployment in geo-distributed settings. To replicate realistic network delays, we extract ping statistics for 32 cities worldwide from WonderNetwork [116] and assign each validator to a city in a round-robin fashion.

Figure 2.5 shows the distribution of transaction latencies of systems deployed in one data center with unmodified network settings and when deployed with geo-distributed settings. For each experiment run, we use 32 validators and fix the transaction load to 64 tx/s, which all evaluated systems can handle without congestion. Except for Algorand and Stellar, network delays visibly impact transaction latencies.

The effect is the most pronounced for the permissoned systems Diem and Hyperledger Fabric. This suggests that these systems can only operate in controlled/closed network environments, as any change in the network significantly affects the performance. We also observe a significant impact of network conditions on Avalance. This is due to the poll-based nature of metastable consensus. The poll rounds are visible in Figure 2.5.

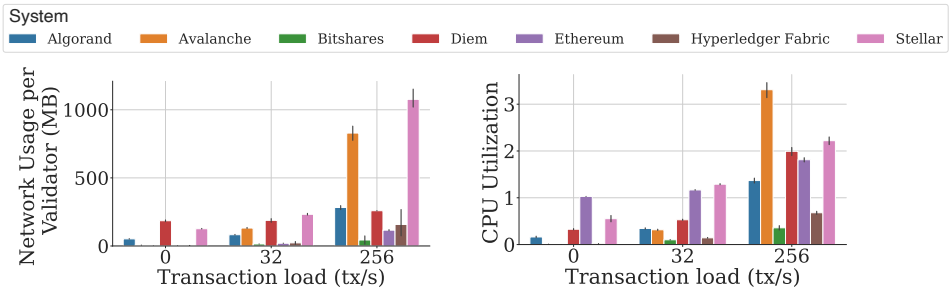
For BitShares, we observe that validators occasionally fail to produce a block with higher network latency. For Diem, consensus progress stales a few seconds after starting the experiment, and rounds are timing out without confirming any transaction, violating system liveness.

2.5.5 Network and CPU Utilization

Finding 6. *Algorand, Stellar, Diem show high network utilization under idle load. Ethereum, Stellar and Diem consume significant CPU resources under idle load.*

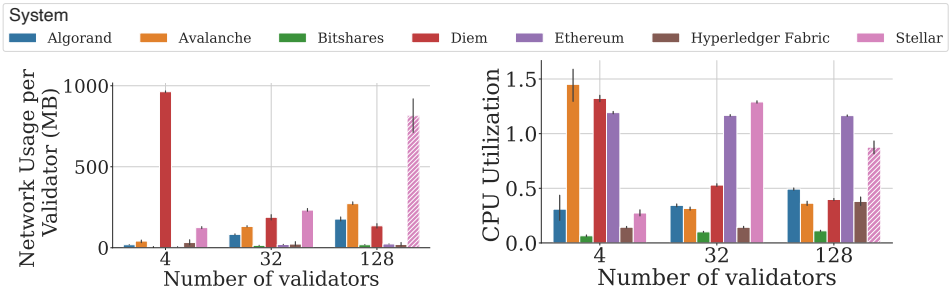
We track the total network usage (inbound and outbound traffic) and average CPU utilization for each system while increasing λ and fixing n to 32. To obtain insights into the system under idle load, we also run blockchain systems with $\lambda = 0$ tx/s. Figure 2.6a shows that the network usage per validator quickly grows for Avalanche and Stellar as the transaction load increases. For $\lambda = 256$ tx/s, both Avalanche and Stellar use over 800 MB of network traffic per validator process. BitShares is the most network-efficient, using only 80 MB per validator for $\lambda = 256$ tx/s. We also observe that Algorand, Diem, and Stellar

[1] TC documentation: `https://www.linux.org/docs/man8/tc-netem.html`



(a) Network usage (b) CPU utilization

Figure 2.6: The resource utilization of the evaluated systems with increase in transaction load ($n = 32$).



(a) Network usage (b) CPU utilization

Figure 2.7: The network and CPU usage for evaluated systems, when increasing the number of validators ($\lambda = 32$ tx/s).

show 10x to 100x more bandwidth consumption under no transaction load compared to other systems.

Figure 2.6b shows the average CPU utilization when increasing λ. The mining process of Ethereum is continuously utilizing a single CPU. Under $\lambda = 256$ tx/s, BitShares and Hyperledger Fabric are the most CPU-efficient compared to other systems. Our synthetic workload results are consistent with real-world observations of liveness issues for Avalanche and Stellar [147, 148].

Figure figure 2.7a shows that Diem is using significant network resources for $n = 4$: 970 MB per validator process. This number decreases quickly when n increases. We explain this behavior by the self-adjusting round times of the Diem consensus mechanism: as n increases, rounds take longer to complete, lowering the bandwidth usage. For Stellar, we see the opposite effect: network usage becomes significant for $n = 128$. Although Stellar cannot process transactions for $n = 128$, we report its resource usage nonetheless. Inspection of Stellar logs reveals that validators lose track of consensus, clogging the network with resynchronization messages.

Figure figure 2.7b shows how CPU utilization behaves when increasing n. BitShares is the most CPU-efficient for all evaluated values of n. The CPU load of validators decreases for Avalanche and Diem as n increases. Since we fix the transaction load, adding more

validators decreases the individual load.

2.6 Related Work

Blockchain benchmarking and its associated challenges has received attention from other researchers. Fan et al. present an extensive survey outlining methods for evaluating blockchain performance [118]. The work of Wang and Ye describes benchmarking tools and consensus mechanisms and outlines techniques to improve the throughput of blockchains [149]. The authors of these studies summarize work on blockchain performance but do not conduct benchmarking studies themselves.

Popular blockchain benchmarking tools are Blockbench [122], Hyperledger Caliper [123], and DAGBench [124]. Blockbench, introduced in 2017, is the earliest benchmarking framework for blockchain and is specifically designed to evaluate permissioned blockchains [122]. Blockbench measures the performance of components commonly found in blockchains, e.g., the transaction execution engine). The authors of Blockbench evaluate the performance of Hyperledger Fabric, Ethereum, and Parity. Hyperledger Caliper is a benchmarking tool for the performance evaluation of specific systems with a set of pre-defined use cases [123]. Hyperledger Caliper primarily supports projects by the Hyperledger Foundation. DAGBench is a benchmarking tool for DAG-based blockchains [124]. The authors evaluate three popular DAG-based blockchain implementations [150]: IOTA, Nano, and Byteball. However, DAGBench does not allow for a comparison of other blockchains. The authors of BCTMark [129]. present an Ethereum performance evaluation.

2.7 Conclusion

We have presented Gromit, a generic benchmarking framework for blockchain solutions. By treating each blockchain system as a transaction fabric, our framework enables any blockchain system's integration, benchmarking, and performance analysis. We leverage the functionalities of Gromit and conduct the largest blockchain benchmark to date, involving seven prominent blockchain systems. Our main finding is that none of the evaluated solutions can handle beyond 1'000 transactions per second as the number of validators increases. Yet, they show relatively low transaction latencies on average. We also find that synthetic workloads can provide accurate predictions of performance limitations, as our findings are consistent with real-world observations on performance incidents on Avalanche and Stellar networks.

3

LØ: An Accountable Mempool for MEV Resistance

Possible manipulation of user transactions by miners in a permissionless blockchain systems is a growing concern. This problem is a pervasive and systemic issue, known as Miner Extractable Value (MEV), incurs highs costs on users of decentralised applications. Furthermore, transaction manipulations create other issues in blockchain systems such as congestion, higher fees, and system instability. Detecting transaction manipulations is difficult, even though it is known that they originate from the pre-consensus phase of transaction selection for a block building, at the base layer of blockchain protocols. In this paper we summarize known transaction manipulation attacks.

We then present LØ, an accountable base layer protocol specifically designed to detect and mitigate transaction manipulations. LØ is built around accurate detection of transaction manipulations and assignment of blame at the granularity of a single mining node. LØ forces miners to log all the transactions they receive into a secure mempool data structure and to process them in a verifiable manner. Overall, LØ quickly and efficiently detects reordering, injection or censorship attempts. Our performance evaluation shows that LØ is also practical and only introduces a marginal performance overhead.

3.1 Introduction

Enabled by blockchain technologies, Decentralised Finance (DeFi) tools and mechanisms have generated a lot of interest as building blocks for novel digital markets, both in terms of practical applications amounting to over 80 billion USD total value locked at the moment of writing, and in terms of significant research interest [151]. Furthermore, these tools enable monetization mechanisms for the new paradigm of Web3 development, providing alternatives to monopolistic centralised digital platforms. Decentralised exchanges, lending markets, derivatives, and other products built on permissionless blockchains are just some examples of these novel financial applications. However, these developments, are undermined by unresolved issues of transaction manipulations, such as *censorship, injection*, and *re-ordering* of transactions, at the expense of application users at underlying layers of blockchain protocols.

This problem led to the notion of Miner Extractable Value (MEV)[1], which refers to the maximum revenue a miner can obtain from benign or manipulative transaction selection for block production [88, 152]. This problem is a pervasive and systemic issue at large scale as exemplified by the Ethereum blockchain, where MEV transaction manipulations have generated over 320 USD million of revenue for bots and miners [153]. Furthermore, over 90% of the blocks produced on Ethereum contain MEV transactions [152]. Such manipulations not only undermine users' trust, but also induce systemic issues like congestion, inflated fees, and system instability [154].

We argue that the root cause of MEV is a lack of accountability at the base layer of permissionless blockchain protocols, sometimes referred to as 'dark forest' [152]. By base layer, we refer to the processing steps that happen before consensus has to be reached on a block, such as sharing pending transactions (recorded in the mempool) with other miners and assembling them into blocks. In contrast to what happens at the consensus layer, at the base layer miners are expected to act as trusted parties. As such, a miner that creates a new block can arbitrarily select the transactions from its mempool. In practice, miners can therefore arbitrarily censor, inject or reorder transactions [155].

While this problem has received certain attention in the context of MEV mitigation tools, there are no comprehensive solutions preventing these types of transaction manipulations [156]. Most of the proposed solutions in this category focus on the application layer and on consensus layer mitigation tools [157]. Many of these tools do not prevent MEV attacks but rather aim to mitigate them. The most well known approach, Proposer Builder Separation (PBS) [158], is implemented with the Flashbots middleware on Ethereum and does not prevent MEV, but only the redistribution of its associated revenues. Some proposed theoretical solutions, such as fair ordering consensus protocols [159], prevent transaction manipulations. However, these algorithms assume permissioned settings and small network sizes, and require important modifications of the blockchain consensus layer.

As transaction manipulations arise from the lack of accountability at the base layer of blockchain protocols, we argue that comprehensive mitigation of MEV requires addressing trust assumptions at this particular layer. To address them, we design LØ, an *accountable mempool* protocol.

[1] Sometimes also referred to as Blockchain Extractable Value, or Maximum Extractable Value.

In LØ miners become accountable for the process of transaction selection and ordering. As new transactions are propagated among miners, they exchange and record commitments on the content of their mempools with each other. New transactions are shared in bundles, and commitment is recorded on a whole transaction bundle. This provides a local partial ordering of transactions. Our system is based on *pairwise commitments* that are exchanged during a mempool reconciliation phase, which is executed before the consensus protocol. This allows miners to witness each others' transaction selection and commit to a particular order and set of transaction that they will use for block generation. Therefore, LØ ensures that any transaction manipulation, such as transaction censorship, injection and reordering, can be detected and proven by a correct node.

Our system is agnostic to a specific type of consensus protocol in a permissionless blockchain system. It can be seamlessly integrated with existing blockchain solutions, as a relatively simple modification of a Peer-to-Peer (P2P) protocols that propagates transactions and blocks. In addition, it does not require any additional cryptographic setups, and it does not impose a significant performance overhead. We leverage Minisketch data structure for the reconciliation of mempools to implement bandwidth-efficient commitments [160].

This chapter makes the following contributions:

• We identify key types of transaction manipulation attack primitives at the base layer. We propose a new taxonomy based on these attack primitives that can grasp all potential MEV attacks. We discuss the stages of the transaction processing pipeline that allow for these manipulations by miners. (§3.2).

• After describing our system model (§3.3), we provide an overview of LØ, an accountable base layer protocol specifically designed to prevent transaction manipulations. LØ is built around accurate detection of transaction manipulations and assignment of blame at the granularity of a single mining node. We discuss specific policies targeted at the detection of different MEV manipulations in (§3.4).

• We detail how LØ detects transaction manipulation attacks and potential mechanisms for the enforcement of these policies (§3.5). We further discuss possible attacks against accountability in LØ.

• We present our performance evaluation, which demonstrates that LØ is practical. It is both bandwidth and memory efficient. For example, it only requires up to 10 MB of additional storage for a network of 10,000 nodes and a workload of 20 transactions per second. At the same time, it is at least four times more efficient than the classical flooding-based mempool exchanges (§3.6).

3.2 Transaction Manipulations at the Base Layer

We distinguish the *base layer* of a blockchain system from its consensus layer. In the complete life-cycle of a transaction from its creation to its inclusion in a blockchain, the base layer corresponds to the steps that precedes the block consensus phase as illustrated in Fig. 3.1. These steps include the creation of the transaction and its initial sharing, its inclusion in the mempools, the reconciliation of the mempools between miners, and the inclusion of the transaction in a candidate block.

We emphasize that the block-building phase, where a miner selects transactions that it includes in a candidate block, is a pre-consensus phase. Indeed, while sometimes block

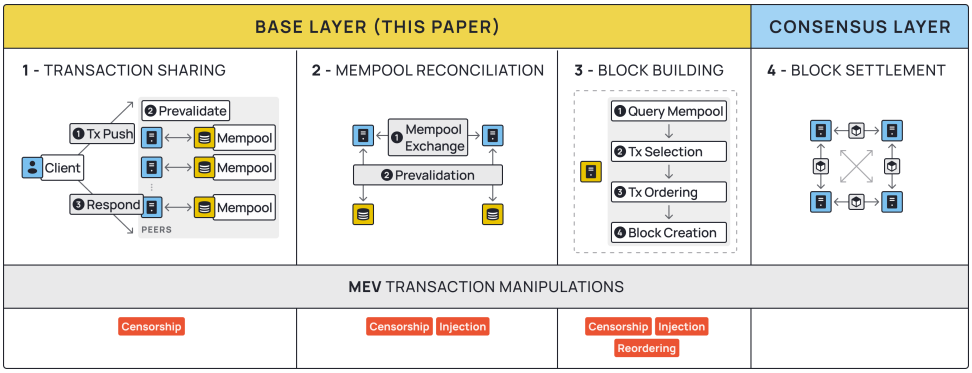


Figure 3.1: Functional modules of a permissionless Blockchain divided into two layers (base and consensus) and four stages.

building is described as part of blockchain protocols, it is strictly speaking not a part of the consensus mechanism as blocks can be produced offline, as illustrated by PBS in Ethereum and selfish mining in Bitcoin [39]. We further distinguish the base layer from the network layer of blockchain protocols, as the latter is required in all transaction processing phases, including during consensus.

The base layer typically provides much lower guarantees against misbehaving nodes than the consensus layer. Miners only conduct checks on the validity and priority of transactions (which is related to miners fee) and add it to a local pool of unconfirmed transactions referred to as the 'mempool' [161]. However, miners are considered to be trusted parties with regard to the selection, withholding, and ordering of transactions [155]. Therefore, all phases of the transaction life-cycle that precede consensus allow transaction manipulations.

3.2.1 Transaction Manipulation Primitives

We consider practical attacks that include reordering of transactions by miners. These attacks have been observed in practical settings and described in academic works that relate to MEV [157]. In practice, these attacks combine different types of transaction ordering manipulations. A common taxonomy of MEV attacks is application-specific and depends on the source of attack revenue. Well known attack types include *sandwich-attacks, front running, back running*, which are associated with decentralized exchanges, *sniping*, which is associated with Non Fungible Token auctions, and *liquidations*, which are associated with collateralized loan protocols. This taxonomy evolves as new MEV attacks rapidly emerge with new applications.

In this paper we consider a different taxonomy focusing on specific attack primitives on the base layer. These primitives allow a broad range of MEV, either on their own or in combination. Namely: *censorship, injection*, and *re-ordering* of transactions.

Censorship. Censorship is the ability of a miner to delay or ignore new transactions. Censorship can enable different financially motivated MEV attacks, such as *sniping*, executed alone or in combination with other primitives. For example, when receiving transactions for a bid in Non Fungible Token auctions, a faulty miner can censor competing

transactions to become the auction winner. This censorship mechanism can take place either during the mempool inclusion phase, or during the block inclusion phase.

Mempool Censorship. Faulty miners can ignore transactions received from some other nodes, and exclude their valid transactions from their mempool. We assume that a faulty miner either provides a fake transaction reception acknowledgment, or does not acknowledge it at all. This type of attack enables censorship at the level of a mempool [157], and facilitates transaction manipulation based on front-running.

Blockspace censorship. Faulty miners can exclude valid transactions from blocks, even after acknowledging their reception and including it in the mempool. This enables transaction censorship at the level of blockspace.

Injections. We assume that honest miners include transactions received from other nodes in new blocks in a deterministic order. Honest miners can also add their own new transactions, under the assumption that updated mempool commitment is shared with other nodes and acknowledged. Faulty miners inject new transactions in blocks in an arbitrary manner, without prior sharing of the updated mempool and without acknowledgements. This type of attack can result in certain types of transaction manipulations such as *front-running, sandwich, back-running* [88].

Reordering. Faulty miners can reorder transactions in a mempool and in a block in a way that deviates from a protocol and violates expectations of other nodes. Reordering is different from an injection attack, since a faulty miner does not add new transactions itself, but manipulates the order of transaction received from other nodes.

3.2.2 Transaction Processing Stages

Attacks can happen at different stages of the transaction life-cycle. We model the processing of a transaction in a generic blockchain system in Fig. 3.1. This processing happens in four stages: (I) initial transaction sharing, (II) mempool reconciliation, (III) block building and (IV) block settlement. In the following we describe each stage, and discuss the corresponding attacks that enable transaction manipulations such as MEV.

Stage I. Initial transaction sharing. A transaction is first created at the client side. The client signs the transaction with its private key. The transaction contains all the required context to be processed by miners, such as signature, UTxO address, execution commands, transaction fee, etc. The client shares the transaction with a subset of peers that it personally knows or whose identity is publicly known (step ①). The peers receive the transaction and attempt to prevalidate it (step ②). Our system is agnostic in respect to the choice of specific consensus protocol which will define the requirements for transaction prevalidation. For example, successful prevalidation of a transaction may require: valid signature from a client, sufficient amount of in a client account, and inclusion of a sufficient transaction processing fee.

Miners that successfully prevalidate a transaction insert it in their local mempool storage. Optionally, miners might respond to the client with the transaction status, to acknowledge inclusion of a transaction in a mempool (step ③). Also optionally, client can query a miner to get an acknowledging of transaction inclusion in a mempool. A malicious peer can censor the transaction at the point of prevalidation, without adding it to the mempool performing *Exclusion From Mempool*. For example, a peer can exclude a client based on its id, e.g. all transaction originating from a specific address. At the same time,

client and peer can collude to include an invalid transaction into a mempool.

Stage II. Mempool reconciliation. At this stage peers share their transaction mempools (step ①). Typically, a mempool exchange is implemented to first share the transaction ids, and only later selectively share the transaction content of the corresponding ids. Once a miner receives the transaction content, it prevalidates the transaction (step ②), similarly to stage I. In theory, this stage allows miner to converge to the same transaction set for any peer-to-peer network. Unfortunately, in practice, there is no guarantee that miners will converge. Client can be partially or completely excluded from learning particular transactions when communicating with malicious peers. Moreover, miners can inconsistently exchange their mempools. Finally, without a requirement for the mempool reconciliation, malicious miner can exclude or include any transaction without being detected by other miners. Different types of *Injection attacks* and *Exclusion attacks* can be performed by faulty miners at that stage. For example, a malicious miner receiving a high-fee transaction can withhold it from sharing with other nodes in order to include it in own block later.

Stage III. Block building. Upon creating a block, a miner populates it based on information stored in its local mempool data (step ①). For each block, the miner selects a subset of transactions to fill up the blockspace (step ②). The selected transactions are included in the block in a specific order chosen by the miner (step ③). A final block contains additional metadata, like signature, nonce, or timestamp (step ④). Most of the reported MEV is happening at the stage of block building. Indeed, miners can freely select, exclude, or order transactions to maximize their profit, performing *Order manipulation* and *Blockspace censorship*.

Stage IV. Block settlement. LØ is agnostic to the specific consensus process to finalize the blocks. We model miner selection as a random process, where a selected miner build its block and sends it to other miners. The attacks on this stage are extensively discussed in previous works. The most discussed manipulations include block withholding, block reordering and equivocation attacks. We consider the accountability on this stage out of scope. Our solution can be combined with other solutions addressing the manipulations on this stage, such as Polygraph [162].

3.3 System Model

This section describes our system model, which is the classical one for blockchain protocols.

The mining nodes (miners) belong in a set $\Pi = \{p_1, p_2, \dots\}$ and communicate with each other by exchanging messages over the network. We assume that each miner is equipped with a cryptographic key pair, and is uniquely identified by its public key. Nodes have access to a cryptographic signature scheme and messages are authenticated.

Communication Overlay. Nodes form an undirected communication graph that is not assumed to be fully connected. Nodes are free to unilaterally add or drop local connections. Nodes are able to leave and later rejoin the network. Nodes share messages to their overlay neighbors through their direct connections. We use notation N_i to refer to the neighbors of a node p_i, i.e., the nodes that are currently directly connected with it.

Bootstrap and Peer Discovery. We assume that nodes that join the system are able to contact bootstrap nodes that facilitate node discovery. When (re)joining the network, each correct node requests a set of known active nodes from the bootstrap nodes. The bootstrap

nodes are correct, i.e., they serve all nodes and unbiasedly propose a node from a set of locally known. As a result the nodes operate in one network.

Continuous sampling. Correct nodes continuously sample the network through a discovery procedure. LØ is build on top of Byzantine resilient uniform sampling algorithm [163]. Malicious nodes can delay the discovery, however, it is guaranteed that correct node will eventually be able to communicate.

Types of Nodes. In different consensus protocols nodes participating in block creation can be called validators, proposers, builders, etc. Here we only consider the role of block creator and refer to the nodes that create blocks as miners. For the sake of simplicity we do not consider light clients, which our model can trivially cover without modifications.

Miners can create new transactions, and they can also propose new blocks with ordered transaction to be included in the blockchain. All nodes maintain a list of unconfirmed transactions (mempool) and exchange it with other nodes in the network through messages.

3.3.1 Attacker Model

In our network, each node is either correct or faulty. Correct nodes adhere to the reference protocol without data tampering and generate valid messages. Faulty nodes, on the other hand, can deviate arbitrarily from the reference protocol.

We assume that a faulty miner can execute any of the transaction manipulations we previously described: censoring transactions, injecting new transactions out-of-order, or deviating from the canonical transaction order [164]. These attacks can be carried out by a faulty miner in a naive way by sending the same message (e.g., a reordered set of transactions) to all neighboring nodes, or they can attempt to evade detection of manipulations by *equivocating*, i.e., sending conflicting messages to different nodes.

3.3.2 Accountability

We consider the standard accountability property for distributed systems and protocols [87]. We define accountability as the ability to detect transaction manipulations and assign blame at the granularity of a single mining node.

In asynchronous environments, an adversary can try to evade detection as it is challenging to distinguish between a misbehaving node that deliberately ignores requests and a slow node. To circumvent this difficulty, we divide blames into two types: *suspicions* and *exposures*. An exposure is a verifiable proof of misbehavior, while a suspicion is a lack of response to a request.

We consider two desirable properties of accountability:

> **Accuracy**: (1) *Temporal.* No correct node is perpetually suspected by a correct node, and (2) *No false-positives.* No correct node is exposed as misbehaving by other nodes.

> **Completeness**: (1) *Suspicion completeness.* Every misbehaving node that ignores requests is perpetually suspected by all correct nodes. (2) *Exposure completeness.* Given an exposure message on node p_i, every correct node exposes node p_i as misbehaving.

3.4 LØ: Accountable Base Layer

In this section we present LØ which achieves accountability at the base layer. Specifically, LØ is implemented as a modification of mempool reconcilation and block building stages.

3.4.1 New Explicit Policies at the Base Layer

Addressed Manipulation	Current implicit policies	New explicit policies
Censorship	Unreliable Transaction Gossip	Inclusion of All Transactions
Injection	Out-Of-Order Transaction Selection	Transaction Selection in Received Order
Reordering	Arbitrary Order in a Block	Verifiable Canonical Order in a Block

Table 3.1: Implicit policies in the base layer of typical permissonless blockchain and the new explicit policies we replace them with to detect transaction manipulations.

This section introduces LØ, our accountable base layer protocol for permissionless blockchains. LØ improves over the 'vanilla' mempool reconciliation and block building protocols of permissionless blockchains (stages 2 and 3 of Fig. 3.1).

To enable accountability we require to modify some currently implicit or ill-defined polices at the base layer. Our observation is that current implementations of blockchain systems use implicit policies that significantly complicate the detection of transaction manipulations. First, a transaction censorship is not possible to attribute to a miner given an unreliable transaction relay. Every miner has its own relaying policy, and even perfectly correctly behaving nodes may choose not to relay anything at all. Second, miners can build a block with any transactions from the mempool, or even inject new transactions during the block creation. Third, there is no 'canonical order' inside a block, allowing for any type of reordering.

Instead of these ill-defined policies we propose three alternative explicit policies to enable the detection of any transaction manipulations, as presented in Table 3.1. In a nutshell, LØ introduces three new explicit policies: *Inclusion of All Transactions*, *Transaction Selection in Received Order*, and *Verifiable Canonical Order in a Block*. Transaction manipulations are detected as violations of our explicit policies during the mempool reconciliation, or when inspecting the content of a block.

Inclusion of All Transactions. Each miner includes all valid transactions it encountered during the system run in its locally maintained append-only transactions set. Once two nodes are connected they directly exchange their known transactions. The transaction exchange is implemented as a sequence of set reconcilations. The miners exchange multiple transactions in one transaction bundle. This allows two nodes to efficiently obtain the transactions they are missing and as a result end up with the same transaction sets.

The key ability of LØ is that after a successful round of reconciliation both correct nodes are ensured to have a common set of observed transactions. To ensure that none of the transactions is censored and all processed in the same way miners keep all valid transactions they encounter. Miners commit to be able to reveal all transactions they know about, if necessary.

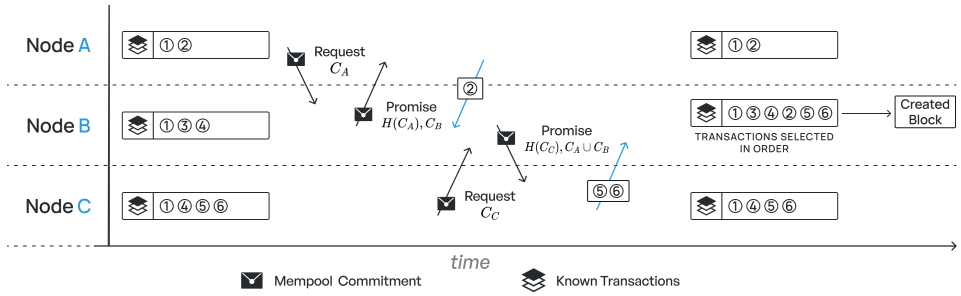


Figure 3.2: Example of a mempool reconciliation in LØ between node B and nodes A, C. Node B preserves the transaction order for the block it eventually creates.

Transaction Selection in Received Order. During the reconciliation process, each miner commits on the order it received a transaction bundle from another miner. To mitigate any out-of-order injections, the miners are required to process the transactions following their insertion order in their mempool. As miners learn and commit on their mempool transactions, the transactions are then naturally ordered according to the order with which they were received.

Verifiable Canonical Order in a Block. Transactions that are inserted into a newly created block are selected according to a deterministic process. In more details, committed transaction bundles are first assembled following sequential order. The order inside a bundle is then pseudo-random: transactions are shuffled using a known shuffling algorithm and an *order seed* value. The order seed value is based on the hash of the last created block.

FIFO is the default order function because it is easy to audit and yields the tightest latency bounds, but LØ is agnostic to *which* deterministic rule is chosen. Concretely, a miner may advertise an alternative *rule identifier* in its commitment (e.g., fee-weighted reservoir sampling, quartile round-robin, or a VRF-based lottery). The rule takes three public inputs—(*committed queue*, *block capacity*, Hash_{n-1})—and must (i) be fully re-computable by every observer and (ii) depend only on data already fixed before Hash_{n-1} becomes known. Any block that deviates from $f_{\text{rule}}(\cdot)$ is provably exposed during block inspection, preserving accountability while giving deployments freedom to optimise for throughput, fee-based incentives, or privacy.

3.4.2 Mempool Reconciliation

The mempool reconciliation process (cf. §3.2.2) forces miners to correctly share the transactions they accepted into their mempool. In practice, LØ's mempool reconciliation uses two techniques: (i) anti-entropy gossip reconciliations [165, 166]; and (ii) signed commitments [160, 167].

Nodes maintain a mempool of all pending transactions and keep a record of all valid transactions they have ever received. Nodes reconcile their mempools to disseminate transactions throughout the system and generate commitments that are exchanged during mempool reconciliations. These commitments cover not only the transactions in the current mempool, but all valid transactions ever received by a node at the time of reconciliation.

Mempool reconciliation serves two purposes: (1) it allows miners to learn about new

3

Algorithm 1 LØ on miner p_i.

```
1:  Ĉ_1,...,Ĉ_N ← ∅,...,∅                          ▷ Last observed commitments
2:  E ← ∅                                          ▷ Set of exposed miners
3:  S ← ∅                                          ▷ Set of suspected miners
4:  procedure NeighborsSync
5:      for p_j ∈ Neighbors(i) do
6:          if C_i \ Ĉ_j ≠ ∅ then                  ▷ Peer j is outdated
7:              S ← S ∪ {p_j}
8:              request C_j ⊇ C_i from p_j
9:          else
10:             S ← S \ {p_j}
11: on Receive C_j
12:     if Ĉ_j ⊂ C_j then
13:         Ĉ_j ← C_j
14:         ΔC_ji ← C_j \ C_i
15:         if ΔC_ji ≠ ∅ then
16:             send H(C_j), C_i to p_j            ▷ Commit
17:         else
18:             send C_i to p_j
19:     if C_j \ Ĉ_j ≠ ∅ ⊊ Ĉ_j \ C_j ≠ ∅) then
20:         E ← E ∪ {p_j}
21:         Broadcast C_j, Ĉ_j
22:
23: on Receive H(C_i), C_j
24:     send txs ∈ ΔC_ij to p_j
25:     Ĉ_j ← C_j ∪ C_i
26:
```

transactions from their neighbors; and (2) it ensures that miners commit to a specific transaction partial order during reconciliation. This partial order must be maintained during block creation. Miners mutually commit to the order by first exchanging a commitment. Miners are inherently motivated to receive transactions from other miners. However, they only disclose the transactions after their counterpart has committed to a specific order of transactions.

Reconciliation Algorithm. In Algorithm 1 we provide LØ's pseudocode for a miner $p_i \in \Pi$. Periodically, miners require their neighbours to commit for new transactions by sending them a request for a new commitment (line 4-10). While the request is pending, the node is suspected. We refer to C_i as a commitment for the set of transactions included by miner p_i. At the same time, the commitment serves as a cryptographic checksum of included mempool transactions.

During the reconciliation process, nodes first exchange a signed commitment C. After receiving the commitments of their neighbours, nodes calculate their transaction set differences with them (line 14). Since the commitment is signed, it can later be used as a proof of inclusion of transactions—any receiver can use the commitment C_j as verifiable evidence that node p_j should have included transactions in its mempool.

Our mempool reconciliation between a miner p_i and miner p_j works in two phases. In the first phase, miner p_i sends to miner p_j a request to commit to new set of transactions (line 8). A peer p_j receives the request and responds either with its new C_j that already includes all transactions (line 18), or with a new commitment fixing locally the order of transactions ΔC_{ij}, i.e., a promise to apply them immediately after all known local transactions C_j (line 16). In the second phase, miner p_i sends all the transactions corresponding to the ΔC_{ij} to peer p_j (lines 23-25).

All miners store at least the last received commitments from their overlay neighbors (line 13). On receiving a checksum C it is first validated against previously received set $\widehat{C}$ (line 19-21). The set $\widehat{C}$ is grow-only and keeps all the transactions committed by the node. If C is inconsistent against the previously reported messages $\widehat{C}$, the evidence of the faulty behavior is shared with other nodes (line 21). This inconsistency could happen for example when a faulty node is trying to hide a previously reported message or does not report a message received from other nodes.

Example. Fig. 3.2 illustrates a possible mempool reconciliation. Nodes A, B, and C first exchange transaction commitments. Note that commitments can also be received indirectly, but this scenario is not included in Fig. 3.2 for simplicity. Node A sends a request, along with the mempool commitment C_A, to node B. Node B reconciles commitment C_A with its own C_B and promises to include node A's missing transactions immediately after all transactions C_B. Node A promptly sends the missing transaction 2 to node B. Shortly afterward, node C reconciles with node B in a similar manner. However, this time, node B promises to include transactions of node C only after the transactions 1,3,4,2. Let's assume that later, node B creates a new block, possibly because it is elected as a consensus leader. Node B must then select all transactions in the order of the commitment it made, which is 1,3,4,2,5,6.

Implementation Details. LØ employs *Minisketch* and *Bloom Clocks* to implement the mempool reconciliation protocol efficiently. A commitment in this context includes both the miner's Bloom Clock and Minisketch. These data structures serve two primary

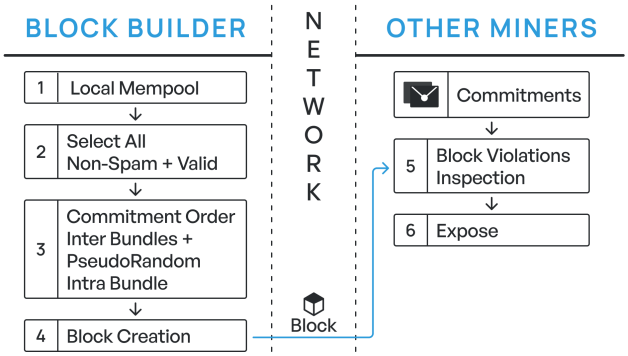


Figure 3.3: Block building and inspection in LØ.

purposes: (1) they identify inconsistencies with the digests shared in previous rounds, and (2) they facilitate set reconciliation to identify a miner's unknown transactions.

A **Minisketch** is a data structure proposed for the bandwidth-optimized exchange of transaction sets between nodes in the Bitcoin network [160]. Initially proposed for the reconciliation of mempool data, it can also be used to optimize block propagation. In this protocol, a sketch serves as a "set checksum". The primary advantage of Minisketch is its ability to reconcile quickly and accurately. However, it has a downside: the requirement to decode the reconciled Minisketch, which can fail. In such cases, we repeat the process by dividing the set in half and sending two sketches.

A **Bloom Clock** is a space-efficient, probabilistic data structure used for the partial ordering of events in distributed systems [168]. LØ uses Bloom Clocks to swiftly detect inconsistencies between two sets. In rare cases, when a Bloom Clock fails to detect an inconsistency due to collisions, we resort to a hash checksum. We employ Bloom Clocks to speed up the verification of inconsistencies between two sets.

Summary 1 *The pairwise commitment scheme ensures that miners are committed to all transactions they discover according to the order with which they are received.*

3.4.3 Block Building

To avoid manipulations during the block building stage, we slightly modify the 'vanilla' block building process with our new policies. The modified block-building process is shown in Fig. 3.3.

Transaction Selection. Peers select all transactions they encounter during the mempool reconcilation phase and that are included in the mempool (step 1). Miners must verify these transactions. The transactions that are not valid are not included in the block. The transactions that have fees lower than some threshold are not included in the block, and are rejected (step 2).

Transaction Ordering. The selected transactions are ordered in a verifiable canonical way (step 3). Recall from the mempool reconciliation process that transactions are partially ordered with the commitments order as the commitments define the order between transaction bundles. We also define a deterministic pseudo-random order function inside

each of the bundle. We use a hash of previous block as a seed for the intra-bundle order function.

Block Inspection. Next, a block is created (step 4) and shared with the network. Given the mempool commitments, any node can verify the produced block by inspecting its content with respect to the LØ reference protocol (step 5). Note that block inspection is a separate process than block validation, and does not affect the block inclusion into the chain. Any violation exposes the block creator (step 6), by comparing the block content with the known commitments. Our protocol is agnostic to the specific punishment mechanisms, but we discuss some options in Section 3.5.3.

Summary 2 *During the block building process, miners select and order transactions deterministically.*

3.5 Dealing with Attacks

This section presents an analysis of various attacks and discusses how the integration of detection mechanisms and a broad spectrum of enforcement tools can counter them.

3.5.1 Detection of Transaction Manipulations

Every node utilizes a block inspection module to detect violations. Nodes are required to disclose all their known transactions and they must consistently disclose each commitment or they run the risk of being identified as faulty. An inconsistency is detected when comparing two commitments, provided both sets contain at least one transaction.

Nodes are obligated to respond to commitment requests. Failure to do so results in an eventual fault suspicion by every correct miner in the network. Reconciliation messages and proposed blocks are validated against the protocol rules. Violations, such as censorship of particular transactions, commitment inconsistencies, or message tampering, can then be identified. Evidence of faulty behavior is disseminated across the network by correct miners.

Countering Attacks during Mempool Reconciliation. Every node involved in a mempool reconciliation retains a signed commitment acquired from other nodes, which can be used to identify faulty nodes. Sufficient interaction with correct nodes in the network makes it virtually impossible for a node to manipulate its mempool and not be detected. The mempool reconciliation process thus ensures reliable detection of injection and mempool censorship attacks. A misbehaving miner attempting a front-running attack, for example, may inject a new transaction out-of-order. However, this attack is swiftly detected as the injected transaction would be inconsistent with previous commitments.

Enhancing Detection Resilience. After a mempool reconciliation between two miners, they can mutually detect each other's violations. Throughout the operation of the system, miners collect commitments from all their overlay neighbors. Consequently, an overlay neighbor can detect a violation. However, if an overlay neighbor is offline, it cannot broadcast the exposure message to other miners. To enhance resilience, miners share between each other a sample of the last commitments they received. This allows other non-neighbouring miners to also detect violations.

Countering Attacks during Block Building. The order function ensures that order manipulation attacks can be detected, as any block where the transaction order deviates

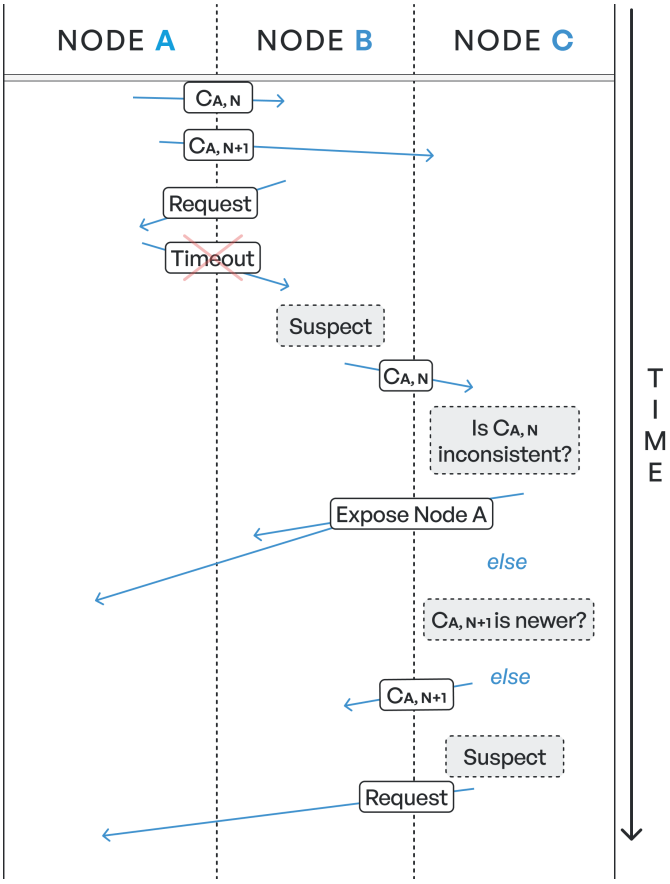


Figure 3.4: Consistency check and suspicion mechanisms.

from the canonical one will be detected. Similarly, a block-space censorship attack is detected as a deviation from the selection function rules.

3.5.2 Suspicion and Misbehavior Sharing

LØ provides guarantees that violation of block production rules can be reliably detected by other nodes in the block inspection process and that misbehaving node will be exposed. Our accountability mechanism provided in LØ that consists of suspicions, equivocation detection, and exposure.

Suspicions. The *Accountability Mechanism* incorporates liveness checks and propagates transaction commitments between nodes through indirect paths. If a node does not respond to transaction requests before a timeout, it is suspected by the requester. The requester may resend the request multiple times before suspecting the node. Correct nodes retain all pending requests. If a node is suspected, the requester broadcasts the suspected requestee's identity to other nodes, along with information on pending requests

and the requestee's last known commitments. A node may retrieve pending requests after a partition or a crash. Once it publicly responds to all pending requests, no correct node will suspect it.

In Fig. 3.4, node B has an earlier commitment ($C_{A,n}$) from node A. Node C has the latest commitment ($C_{A,n+1}$) from node A. Node B sends a request for a commitment on a particular transaction τ from node A, but does not receive a response. After a timeout, node B suspects node A and broadcasts a suspect status along with the latest commitment ($C_{A,n}$) from A that is available to B to its neighbors, in this case, to node C.

Equivocation Detection. A consistency check occurs when a node is suspected. Commitments are append-only sets and thus follow chronological order. When a node has two commitments from a neighbor, it can easily detect any inconsistency between the previous commitment n and the latest commitment $n+1$ using its bloom clock. Nodes can receive commitments from other nodes both directly and indirectly. Consider an example of suspicion and consistency check in Fig. 3.4. Node C receives two commitments originating from node A, i.e., commitment $(C_A, n+1)$ from node B, and $(C_A, n+1)$ from node A. Node B has tried to get a commitment on transaction τ from A and suspects A because of the high response delay. Node C will check whether (C_A, n) and $(C_A, n+1)$ are consistent with each other.

- If these commitments are inconsistent, node C exposes A as a misbehaving node.
- If (C_A, n) and $(C_A, n+1)$ are consistent and $(C_A, n+1)$ already includes a commitment on a transaction τ, then node C will share the latest commitment $(C_A, n+1)$ with B.
- If (C_A, n) and $(C_A, n+1)$ are consistent but $(C_A, n+1)$ does not include a commitment on τ, then C will send a request for commitment on τ to C and suspect C.

Summary 3 *Any mempool counterpart can submit a proof of misbehavior showing inconsistency between a mempool commitment and a produced block.*

3.5.3 Possible MEV Prevention Mechanisms

Reliable detection and blame assignment allow for MEV mitigation through the enforcement of policies. The choice of specific enforcement mechanisms depends on the consensus protocol. Given that LØ is agnostic to the particular consensus algorithm used, a detailed analysis of specific enforcement mechanisms is beyond the scope of this paper.

For instance, in *Proof-of-Stake* (PoS) consensus algorithms, various slashing strategies can be applied to misbehaving nodes [169]. Since validating nodes in PoS must invest a certain amount of funds to become validators, slashing of stake incurs a financial loss. For consensus algorithms based on the reputation of validating nodes, slashing of reputation can equivalently serve as a penalization mechanism [170]. Misbehaving nodes can also be penalized at the network layer level, such as temporary disconnection from the network [155]. In addition to penalizing misbehaving miners, detection allows the implementation of mechanisms for the rejection of blocks that deviate from the canonical transaction order [171]. However, this latter approach imposes significant trade-offs on the modification of the consensus protocol.

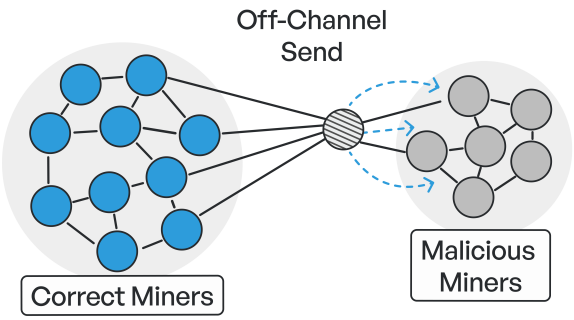


Figure 3.5: Illustration of colluding malicious miners communicating off-channel to evade detection.

3.5.4 Addressing Accountability Attacks

In our model, we assume that miners are incentivized to learn about more transactions. This assumption aligns with empirical observations, as miner profitability correlates with their ability to discover new transactions [156].

In our system, miners only discover transaction content after exchanging commitments. Hence, by learning about new transactions, miners commit themselves to mempool commitments. However, a potential loophole exists for malicious miners. A miner could conspire with an accomplice who does not interact with correct nodes to create a block for them using a manipulated transaction order. This attack is depicted in Fig.3.5. A malicious miner can transfer a transaction, denoted as *Tx*, to another colluding miner or to a Sybil miner under their control. Since a colluding miner has not exchanged commitments with the originator of *Tx*, it can attempt to reorder or inject transactions, and propose an alternative block. However, this type of attack is impractical due to several reasons:

- Colluding miners can only front-run or back-run an entire original transaction bundle. Any attempt to inject, censor, or reorder transactions within a transaction bundle is eventually detectable by correct miners. This significantly restricts the attack granularity, a crucial factor for MEV profitability.

- Colluding miners or Sybils cannot respond to queries from honest miners to evade commitments. They can only learn about new transactions via malicious nodes acting as a bridge. However, such a non-responding set of colluding miners is eventually detected and suspected.

- Colluding miners or Sybil miners must have a high probability of becoming the consensus leader to include a specific transaction. To increase this success rate, a substantial set of colluding miners or Sybils is required, which is costly considering the initial investment and the absence of profits from honest protocol participation.

Finally, to further mitigate the attack, one option is to require sufficient Proof-of-Interaction during block creation. Specifically, the block creator must also include signatures from a sufficient number of miners (based on mining power or stake), thereby proving recent interaction with them.

3.6 Evaluation

This section presents our evaluation of LØ focusing on its resilience against malicious nodes and the impact of such nodes on detection. We also discuss the overhead associated with LØ.

3.6.1 Experimental setup

LØ was evaluated experimentally on a national research cluster [114]. Each server in the cluster is equipped with an Intel Xeon E5-2630 CPU with 24 physical cores operating at 2.4 GHz, hyper-threading enabled, and 128 GiB of main memory. The servers are interconnected via a Gigabit Ethernet network. LØ was implemented in Python. We emulated realistic network latencies using netem[2] and incorporated ping statistics from 32 cities worldwide from the WonderNetwork dataset [116]. Each miner was assigned to a city in a round-robin manner.

Unless otherwise stated, the parameters for the reported experiment were set as follows: The experiment was conducted with 10,000 nodes, generating a workload of 20 transactions per second, with each transaction being 250 bytes in size. The transactions were injected into our system based on a realistic dataset of Ethereum transactions [172]. Each experiment was repeated 10 times, and the average result of these runs is reported.

We constructed a connected topology where each node had eight outgoing connections and up to 125 incoming connections, in line with the default Bitcoin parameters. Every node attempted to reconcile with three random neighbors every second. The request timeout was set to 1 second. If a request was not fulfilled within this time, it was resent three times, after which the node was suspected of being faulty. The Minisketch size was set to 1,000 bytes, sufficient to reconcile a set difference of up to 100 transactions, allowing the Minisketch to fit into a single UDP packet. If reconciliation failed, all transactions were divided into two subsets, and the process was repeated with two sketches. The size of Bloom-Clocks was fixed at 32 cells (i.e., 68 bytes in total).

3.6.2 Resilience

We assess the impact of colluding censoring miners on the network, specifically focusing on their effect on the convergence of correct nodes. In this scenario, malicious miners attempt to prevent correct nodes from learning about transactions, commitments, exposure, and suspicion messages. All malicious miners are assumed to be interconnected. For these experiments, we ensure that the correct nodes remain connected via some path in the network by initially running an unbiased sampling algorithm [163, 173].

Fig. 3.6 illustrates the time required for all correct nodes to converge, depending on the number of faulty nodes in the network. The presence of faulty nodes marginally increases the time needed for all correct nodes to learn about the exposure message, extending it to 6-7 seconds after the first miner detects and creates the message.

We also demonstrate how our system can detect faulty nodes that ignore requests. We report the time until every correct node suspects all faulty nodes (Fig. 3.6, 'Suspicion'). As expected, the time until all faulty nodes are suspected is longer than the time required for

[2]See `https://www.linux.org/docs/man8/tc-netem.html`

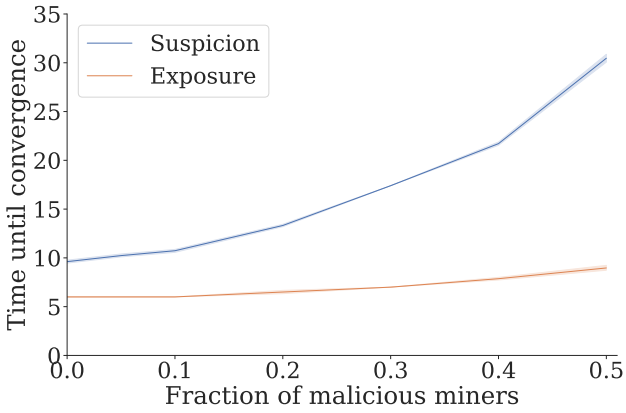


Figure 3.6: Time necessary to suspect or expose a faulty miner depending on the proportion of colluding censoring miners in the system.

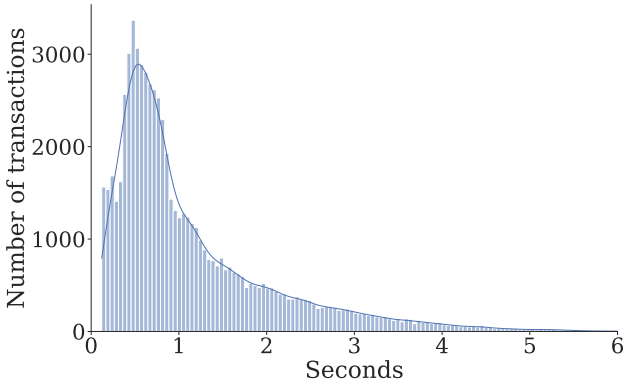


Figure 3.7: Distribution of the time (s) required for a miner to insert a transaction into its mempool.

nodes to discover an exposure message, as the nodes need to submit a request and wait for it to timeout.

3.6.3 Transaction Latency

We report the time necessary for miners to discover a transaction and include it in their mempool. The latency distribution is reported in Fig. 3.7. It appears that all nodes learn about the transaction after contacting 5 to 6 nodes. On average, a transaction is discovered by a node in 1.14 s.

To demonstrate the effects of our new policies on block building, i.e., selecting transactions in order, we simulate a block creation process at randomly selected miners with an average block time of 12 s, which is the block time in Ethereum. We report the average time it takes for a transaction to be included in a block in Fig.3.8. We compare the policy for block creation described in Section 3.4.3 ('Natural' ordering) with the policy that is

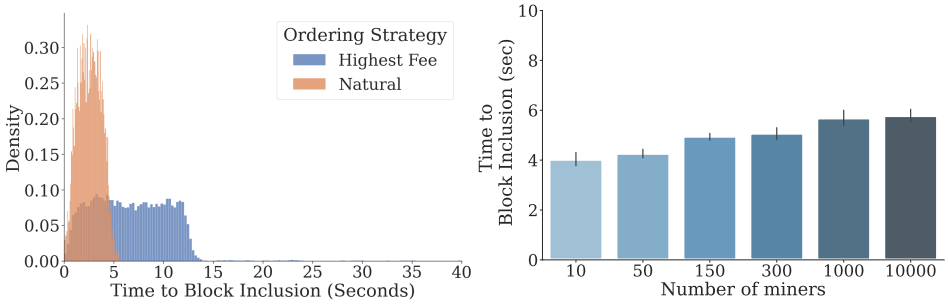


Figure 3.8: (Left) Average time for a transaction to be included in a block for the vanilla algorithm used in Ethereum and for LØ (i.e., 'Natural' ordering), and (Right) time for a transaction to be included in a block with LØ depending on the system's size.

currently widely used in public blockchains, i.e., creating a block with the highest-fee transactions of the mempool (referred to as Highest Fee').

The average transaction latency for the 'Natural' ordering is 3 seconds, while it is around 7-8 seconds for the 'Highest Fee' strategy. Furthermore, we observe that the 'Highest Fee' strategy exhibits a wide spread along the axes, with many low-fee transactions experiencing very high latency. LØ's orders transactions according to the order with which they have been received by miners, which leads to transactions being processed sequentially and increases fairness.

3.6.4 Protocol Overhead

Bandwidth

We benchmark our protocol against two baseline protocols: 'Flood' and 'PeerReview'. 'Flood' is a traditional mempool exchange protocol where miners initially send a 'Mempool' message containing a list of hashes of the transactions currently in their mempool. The receiving miner compares these hashes against its known transaction IDs and requests any missing transactions.

We also compare LØ to 'PeerReview', a generic accountability protocol that could be used to monitor censorship attempts by miners [87]. Every miner maintains an additional log for each received message. For each miner, we assign 8 witnesses. Periodically, each miner fetches the log from the miners and checks for any injection (commission) or censorship (omission).

The comparison is reported in Fig. 3.9. Note that we omit the bandwidth overhead for sharing transactions, as it is the same for all three protocols. Our protocol is the most bandwidth-efficient compared to the other two protocols, incurring 20 times less bandwidth overhead than PeerReview.

Memory and CPU Overhead

The overhead for encoding and decoding Minisketch scales linearly with the size of the set difference [174]. Minisketch computes a set difference with 1,000 items in 10 seconds. To optimize the usage of the sketch, we hash-partition the mempool space into subsets, as described in [174]. Each time reconciliation fails, the node divides the mempool in half

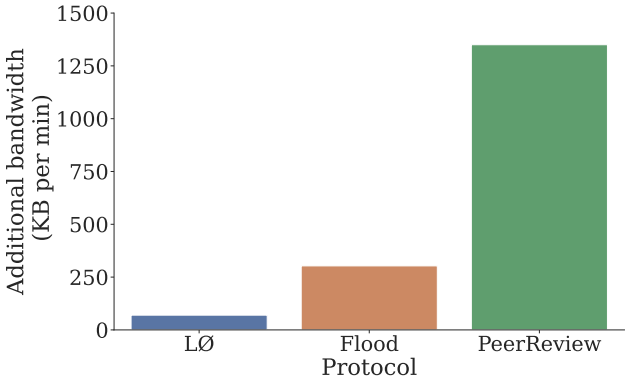


Figure 3.9: Bandwidth overhead measured in KB per minute.

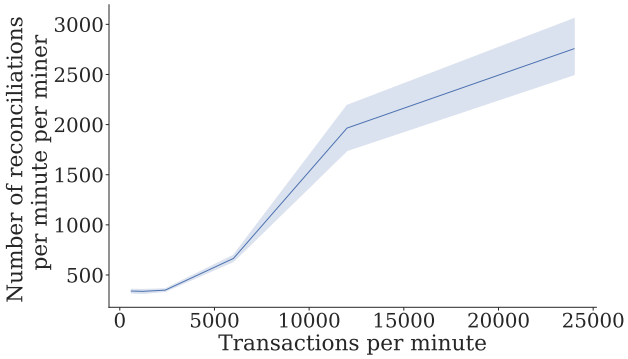


Figure 3.10: Average number of reconciliations per minute depending on the workload (new transactions per minute).

and sends an additional Minisketch for each partition. As a result of this optimization, we encode and decode all sketches required for a set difference of 1,000 items in less than 100 ms. We report the average number of sketch reconciliations per minute per node depending on the workload in Fig. 3.10.

LØ only requires a small additional memory overhead to store the commitments of all its neighbors. The size of the commitment depends on the workload. For example, for a workload of 120 transactions per minute, the commitment size is 1.17 KB, while for a workload of 24,000 transactions per minute, the total size of commitments can reach up to 9.36 KB. Even if the miner stores the commitments of all 10,000 nodes, it would only require 87 MB.

3.7 Related Work

The problem of MEV has attracted a considerable amount of research [152, 157, 164]. Different MEV mitigation mechanisms can be categorized according to implementation at different layers: *application, consensus, base.*

3.7.1 MEV Mitigation at the Application Layer

Decentralized Exchanges aggregators such as Cowswap implement *Batch Auctions* where orders are placed off-chain and not immediately executed, but rather, collected and aggregated to be settled in batches [175].The applications of this approach is tied to a specific application, and thus limited to specific types of MEV attacks (front running and sandwich).

A2MM is a DEX design that atomically performs optimal routing and arbitrage among the considered AMM, minimizing subsequent arbitrage transactions [176].

3.7.2 MEV Mitigation at the Consensus Layer

Proposer-builder separation (PBS) is a proposal aiming at MEV minimization [177].The latest iteration of this mechanism, MEV-Boost, is implemented as a middleware. It enables private communication channels between clients creating new transactions and validating nodes. However, this approach has significant trust assumptions, such as relays not reordering or censoring transactions, which empirically do not hold [157].

Pre-ordering solutions aim to separate transaction ordering from execution to ensure 'fair' ordering. The Helix consensus protocol [178] guarantees random selection and ordering of transactions in blocks, relying on a randomness beacon within the consensus protocol. Aequitas [179] provides guarantees on transaction ordering within a block, but assumes a permissioned environment and introduces significant communication overhead. Pompe [180] is a Byzantine ordered consensus (BOC) protocol that outputs a transaction t and a sequence number s for ordering t. Wendy [181, 182] describes ordering protocols for permissioned systems. Enforcing relative order requires building a dependency graph to prevent transactions from being included in a block before their dependencies [159, 179, 183]. Enforcing fair-ordering is more resource-intensive than enforcing our accountability properties and not practical in a permissionless setting.

Heimbach and Wattenhoffer propose encrypting transaction content, ordering it, and revealing its content only after it has been ordered [184]. This approach is implemented by Fino, which integrates MEV protection into a BFT protocol in the partial synchrony model with a DAG transport protocol [185]. Lyra [186], a Byzantine ordered consensus protocol, also uses a commit-reveal scheme and relies on Verifiable Secret Sharing (VSS). The encrypt-commit-reveal scheme is more resource-intensive than our accountable approach and requires additional trust assumptions to ensure that encrypted transactions are always revealed.

3.7.3 MEV Mitigation at the Base Layer

Secret Mempools hide the content of a transaction so that it cannot be censored, reordered, etc. F3B is a generic approach for online transaction encryption based on a commit-and-reveal architecture [187]. Ferveo is a protocol for Mempool Privacy on BFT consensus blockchains [188]. Both of these solutions assume permissioned settings. Shutter is a frontrunning protection system for Ethereum smart contracts based on threshold cryptography-based distributed key generation (DKG) protocol [189]. It does introduce additional latency and significant trust assumptions, as key generation is delegated to a committee of trusted nodes on a private Tendermint-based blockchain.

ZeroMEV is an existing MEV mitigation solution implemented on the base layer [190]. This solution is Ethereum-specific and implemented on the basis of Geth software fork as

a validator execution client. It orders transactions based on timestamps with local FIFO order. However, this solution does not provide any accountability and requires strong trust assumption as it relies on the altruism of a validator.

3.8 Conclusion

We introduced LØ, an accountable base layer for permissionless blockchains. It is consensus-protocol agnostic and provides detection guarantees for various MEV attacks. LØ mandates that both correct and faulty miners log all received transactions into a secure mempool data structure and exchange and record commitments on their mempool content. Any inconsistency, such as transaction withholding or equivocation, is exposed during a mempool reconciliation process with a correct miner. To ensure the exposure of faulty miners, LØ simply requires correct miners to be interconnected through a network path.

We outlined the transaction manipulation attacks associated with MEV that miners might execute and mapped different attack types to the relevant stages of a transaction's lifecycle within the protocol. Our performance evaluation demonstrates the practicality of LØ. It is bandwidth and memory efficient, using only 10 MB with 10,000 miners and a workload of 20 transactions per second. Moreover, it is at least four times more bandwidth efficient than classical flooding-based mempool exchanges and processes transactions with higher fairness.

4

MeritRank: Sybil Tolerant Reputations for Merit-based Tokenomics

Decentralized reputation systems are emerging as promising mechanisms to enhance the effectiveness of token-based economies. Unlike traditional monetary incentives, these systems reward participants based on the actual value of their contributions to the network. However, the advantages and challenges associated with such systems remain largely unexplored. In this work, we investigate the inherent trade-offs in designing a decentralized reputation system that is simultaneously generalizable, trustless, and Sybil-resistant. Specifically, "generalizable" means that the system can assess various types of contributions across different contexts, "trustless" indicates that it functions without the need for a central authority to oversee reputations, and "Sybil-resistant" refers to its ability to withstand manipulations by fake identities, i.e., Sybil attacks.

We propose MeritRank, a Sybil-tolerant reputation system based on feedback aggregation from participants. Instead of entirely preventing Sybil attacks, our approach effectively limits the benefits that attackers can gain from such strategies. This is achieved by reducing the perceived value of the attacker's and Sybil nodes' contributions through the application of decay mechanisms—specifically, transitivity decay, connectivity decay, and epoch decay. Using a dataset of participant interactions in MakerDAO, we conducted experiments to demonstrate the Sybil tolerance of MeritRank.

4.1 Introduction

Reputation mechanisms in blockchain applications provide numerous desirable properties as system components. These mechanisms can be employed at various layers of blockchain systems. At the infrastructure level, anyone can act as a relay node, competing based on reputation for how quickly and neutrally they distribute transactions [191]. At the protocol layer, Delegated Proof-of-Stake (DPoS) [192] utilizes mechanisms to detect and punish undesirable behaviors, such as double voting on blocks [193]. At the application layer, Decentralized Autonomous Organizations (DAOs) [194–197] employ rewards for participants based on their contributions.

Token-based incentives have emerged as prominent mechanisms in blockchain protocols, addressing several incentivization issues in peer-to-peer networks, such as ensuring network liveness, enhancing network security, and supporting open-source software maintenance. This exploration of incentives has given rise to a new subfield within blockchain design, often referred to as *tokenomics*. Consequently, these mechanisms are integral to most significant blockchain applications, utilized at various abstraction levels [198].

However, the limitations of traditional token-based incentives are increasingly evident through empirical evidence. First, misalignment of incentives often occurs, where monetary rewards can lead to conflicts among participants within complex systems [199, 200]. These conflicts arise because such rewards do not always reflect the actual value of contributions, fostering self-interested behaviors that can undermine collective goals. Second, governance models relying solely on monetary incentives lack the robustness necessary for effective decentralized decision-making [201]. Lastly, these incentives tend to disproportionately benefit larger stakeholders, risking the re-centralization of decentralized networks [202].

An alternative approach is to reward participants based on their contributions or *merits* through a reputation system [203]. Examples of contributions include computational work, proof of bandwidth, proof of storage, participation in DAO governance forums, open-source code development, and discussions in chat messengers. DAOs often establish treasuries to incentivize contributions by periodically rewarding the most active participants with tokens. However, this approach faces significant challenges due to Sybil attacks [96, 204], where attackers use multiple fake identities to manipulate reputations and deplete the treasury. Such attacks are a major obstacle to deploying advanced reputation systems intended to surpass the simplistic economic models often found in current tokenomics discussions [205, 206].

While some previous work has proposed Sybil-resistant reputation algorithms [49, 207, 208], these solutions only partially mitigate Sybil attacks and remain vulnerable to reputation manipulation. First, Sybil identities are often indistinguishable from legitimate identities [209], making many Sybil-detection algorithms ineffective. Second, achieving full Sybil resistance is challenging because Sybil identities can still receive positive feedback from legitimate participants. This positive feedback allows Sybil identities to blend in and accumulate reputation, thereby making Sybil attacks beneficial. Such attacks are prevalent in social networks where interactions and feedback can be easily manipulated [210].

We observe that Sybil attacks become advantageous when the attacker makes only minimal contributions, yet both the attacker and their Sybil identities receive positive reputations and token rewards. This dynamic can undermine the integrity of reputation systems.

Rather than trying to fully prevent Sybil attacks or detect Sybil identities, our approach focuses on limiting the benefits that attackers can gain from such attacks, thereby making them less attractive and practical. To achieve this, we propose MeritRank, a Sybil-tolerant, aggregated feedback reputations compatible with token-based incentive systems. MeritRank achieves Sybil-Tolerance by bounding the benefit of the attack through the use of personalized reputation and decay heuristics to reduce the perceived value of attacker's contributions.

In MeritRank, we introduce and test three types of decay mechanisms, building upon findings from previous research [208, 211], to enhance the resilience of reputation systems against Sybil attacks. *A seed node* is a trusted starting point from which reputations are calculated, serving as a reference for evaluating other nodes' contributions. The three decays are *transitivity decay*, *connectivity decay*, and *epoch decay*. Transitivity decay reduces the perceived value of contributions based on their distance from the seed node. Connectivity decay lowers the value of contributions from nodes that rely on a single path or a limited set of connections to the seed node, such as those connected only through bridges. Epoch decay decreases the value of older contributions. These mechanisms reduce the effectiveness of Sybil attacks, making them less attractive. However, this comes at a cost: honest participants might receive lower reputations due to the decay of their perceived contributions. Despite this drawback, we will see that the improved resilience of the system justifies the trade-off.

This chapter makes both theoretical and practical contributions, which are as follows:

- We analyze and formulate the general trade-offs between desirable properties of reputation in decentralized settings by presenting a *decentralized reputation trilemma* in Section 4.2.

- We present the formalization of MeritRank, along with three types of decay mechanisms—transitivity decay, connectivity decay, and epoch decay—designed to achieve tolerance against Sybil attacks in Section 4.5. We evaluate MeritRank using a dataset spanning over 150 weeks of user interactions within the leading decentralized autonomous organization, MakerDAO, as discussed in Section 4.6. Our experiments demonstrate that transitivity decay and connectivity decay significantly enhance the Sybil tolerance of reputation algorithms. However, we observe that the popular heuristic, epoch decay, used in other reputation mechanisms [197], does not improve Sybil tolerance.

4.2 Background and Related Work

Reputation systems in decentralized environments have been proposed for various applications, and as general models in peer-to-peer systems [211–216]. Accordingly, the limitations of these solutions are relatively well-understood. Some of these limitations include scalability [215], contextual accuracy [214], reliance on partially trusted setups [214, 215], vulnerability to misreporting attacks [217–221], and privacy trade-offs [216].

However, there is an identifiable research gap regarding the general trade-offs inherent to any reputation system implemented in decentralized environments. We are able to identify only a small number of surveys on reputation solutions in decentralized

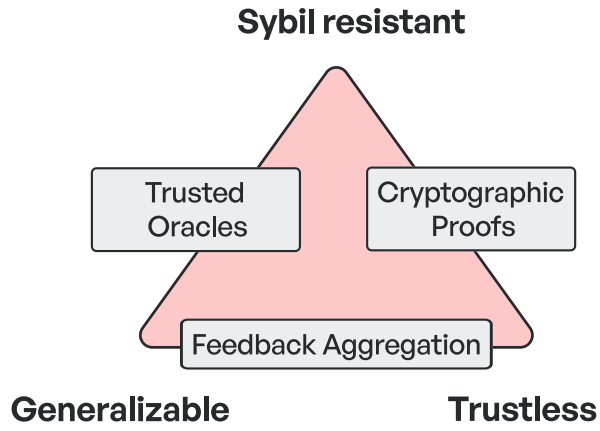


Figure 4.1: The "Decentralized Reputation Trilemma" illustrates the irreconcilability of three desirable properties: Generalizable, Sybil-resistant, and Trustless. The triangle's edges represent approaches that sacrifice one property: Trusted Oracles sacrifice Trustlessness, Cryptographic Proofs sacrifice Generalizability, and Feedback Aggregation sacrifices Sybil resistance.

settings [214, 215], with limited comparative trade-off analysis [216]. The absence of engineering research on reputation in the context of tokenomics and DAOs can be attributed to the novelty of these problems [222], which have only recently gained attention in the academic and engineering communities.

4.2.1 Decentralized Reputation Trilemma

We formulate the inherent trade-offs in decentralized reputation systems as a conjecture on the irreconcilability of three desirable properties. This trilemma is depicted in Figure 4.1. A reputation system cannot simultaneously embody all three attributes: *Generalizability*, *Sybil resistance*, and *Trustlessness*.

Generalizability refers to the system's ability to evaluate and fairly assess a wide variety of contributions, including both technical and human-centric activities. This flexibility allows the system to adapt to different types of participant roles and behaviors, without being limited to specific, easily measurable tasks. A generalizable system must also scale efficiently, maintaining performance as the network grows, while providing accurate, context-aware reputation scores that account for the nuances of each contribution.

A fully *Sybil-resistant* system prevents attackers from manipulating reputation metrics through the creation and control of multiple fake identities [96]. This is crucial for maintaining the integrity of the reputation system, as it ensures that reputation scores genuinely reflect legitimate contributions.

The *Trustless* property means that the system's reputation accounting and evaluation processes do not rely on any single trusted entity. Instead, it leverages decentralized protocols and mechanisms to ensure that all operations are transparent, verifiable, and resistant to tampering or biased influence. This decentralization enhances the system's security and fairness by minimizing the risk of corruption or failure associated with relying

on trusted intermediaries.

These three properties form a trilemma because optimizing for two typically compromises the third. To better understand how different approaches navigate these trade-offs, we examine three primary methods used in decentralized reputation systems: Trusted Oracles, Cryptographic Proofs, and Feedback Aggregation.

Trusted Oracles. This method employs trusted oracles to monitor and calculate reputation scores, relying on predefined reputation functions and participant actions as inputs. Such a system is inherently generalizable, as oracles can process various inputs to determine reputation scores [215]. The challenge of Sybil resistance is managed by entrusting oracles with the task of detecting and verifying the identities of participants, thereby preventing Sybil attacks [223]. However, this reliance on oracles compromises the trustless nature of the system. Participants must trust the oracles to accurately process inputs and honestly calculate reputation scores. Moreover, the dependence on oracles introduces potential points of failure that adversaries could exploit, as observed with price oracles in decentralized finance systems [224].

Efforts to incorporate trustless characteristics within oracles have led to the creation of peer prediction markets. In these systems, participants estimate and report outcomes, with their rewards determined by how well their reports align with those of others. This method is designed to be self-regulating and decentralized, reducing reliance on a single authoritative oracle. However, in scenarios where quick or highly specialized assessments are needed, the peer prediction method might struggle to provide accurate and timely evaluations, thus limiting its generalizability [225].

Cryptographic Proofs. With this approach, peers document their own and others' reputation scores by generating cryptographic proofs of contributions, which are then disseminated across the network. Upon validation of these proofs, participants adjust their reputation scores accordingly. The requirement for verifiable proof of contribution inherently reduces the feasibility of Sybil attacks, as each entity must substantiate its contributions through cryptographic evidence. This method also achieves trustlessness, as it eliminates the need for a singular authoritative body overseeing reputation accounting.

However, the applicability of cryptographic proofs to reputation systems faces significant constraints, particularly regarding generalizability. While certain contributions—such as computational work, proof of bandwidth, or proof of storage—are well-suited for cryptographic validation, this framework struggles to accommodate a broader spectrum of collaborative efforts and human-centric contributions. Many types of cooperative work cannot be easily verified cryptographically, limiting the range of activities that can be effectively accounted for in such a system. Moreover, even when cryptographic proofing is conceptually applicable, its practical implementation faces challenges related to scalability. The extensive overhead required for generating, distributing, and validating cryptographic proofs can significantly strain system resources, impacting both scalability and efficiency [226].

Feedback Aggregation calculates an individual's reputation by collecting feedback directly from other participants on the perceived value of that individual's contributions. The system then aggregates this feedback to determine the overall reputation score. This method is generalizable and trustless. By facilitating peer-to-peer feedback, the system can adapt to diverse application-specific scenarios, allowing participants to provide nuanced,

context-specific feedback on a wide array of interactions or contributions. This adaptability makes it exceptionally suitable for environments where direct peer evaluation is possible and relevant. Additionally, unlike approaches reliant on cryptographic proofs, feedback aggregation does not incur significant overhead costs, making it a more resource-efficient option.

However, this method is susceptible to manipulation, particularly through Sybil attacks. In such attacks, a malicious entity creates multiple fake identities to flood the system with false feedback or fraudulent contributions, thereby distorting the reputation scores [96].

Previous research has proposed various approaches to Sybil-resistant reputation systems. For instance, EigenTrust [227] uses a global trust value computed through iterative aggregation of local trust scores but is susceptible to Sybil attacks if attackers can accumulate sufficient local trust. SybilGuard [228] and SybilLimit [229] leverage social network structures to limit the number of Sybil nodes accepted but rely on the assumption that the social graph is fast-mixing, which may not hold in all cases.

4.2.2 Our Approach to Solve the Reputation Trilemma

Observations based on the reputation trilemma necessitate a solution for reputation systems in decentralized environments that does not completely sacrifice one of the corners of this triangle. Because of the lack of generalizability of *Cryptographic Proofs* and the trust assumptions inherent in *Trusted Oracles*, the *Feedback Aggregation* approach emerges as a viable direction. This is contingent on our ability to improve the resistance of reputation aggregation functions to Sybil attacks.

A common way to achieve Sybil resistance is to emulate a closed system trying to achieve *Sybil prevention*. For example, by requiring participants to undergo identity verification processes, the system can limit the creation of fake identities. However, this approach often conflicts with the principles of privacy and decentralization inherent in open systems. In the context of open, permissionless systems, strict Sybil resistance is not entirely achievable [209]. An alternative approach to Sybil resistance is based on *Sybil detection* [230] and the subsequent exclusion of Sybils from the system. The effectiveness of Sybil detection is generally constrained by several factors. In open, decentralized, and pseudonymous networks, identities are easily created, and there is no reliable way to link digital identities to unique real-world entities. This makes it inherently challenging to distinguish between legitimate users and Sybil identities. Attackers can create multiple identities that behave indistinguishably from honest nodes, making detection algorithms ineffective [209]. Additionally, imposing strict verification measures conflicts with the principles of decentralization and user privacy.

Sybil tolerance [219] focuses on minimizing the impact of Sybil identities rather than attempting to identify and remove them. By accepting that some Sybil identities may infiltrate the system, the focus shifts to reducing the damage they can cause. This can be achieved by implementing mechanisms that restrict the influence any single participant or group of participants can exert on the reputation system. Sybil tolerance is achieved by limiting the relative benefit that an attacker can gain through the use of Sybils. Specifically, a reputation system is considered Sybil-tolerant if, even as an attacker creates additional Sybil identities, the cumulative influence of these identities on the reputation scores remains limited.

One major limitation of Feedback Aggregation systems is their reliance on heuristic methods to determine the influence of participants, which can result in inaccurate reputation assessments. If the heuristics are too strict, legitimate users may be unfairly penalized, reducing the system's fairness and overall effectiveness. Conversely, if the heuristics are too lenient, they may fail to sufficiently limit the influence of Sybil identities, allowing attackers to cause significant harm. Despite these inherent challenges and limitations, we adopt the Feedback Aggregation approach. Our goal is to strike a balance between overly strict and overly lenient heuristics, ensuring fairness for legitimate users while maintaining resilience against Sybil attacks.

4.3 Merit-based Tokenomics

In this section, we present a general system model for merit-based tokenomics. This model describes a reward system for the participants of a generic DAO, where peers provide feedback to each other resulting in a reputation ranking that can be used to distribute token rewards from a DAO treasury proportionally to accrued reputation. We acknowledge that reputation mechanisms in distributed systems face challenges, such as incompleteness of information about peer interactions and peer discovery. Therefore, our design accommodates these limitations by functioning effectively with only partial information.

The model is illustrated in Figure 4.2 and includes four mechanisms. The *accounting mechanism* records locally computed or recorded feedback in a personal ledger. The *gossip mechanism* distributes this information to peers, resulting in the collection of indirect feedback (i.e., feedback gathered from other participants) in the form of a feedback graph. This graph is used by the *reputation mechanism* to calculate participants' reputations. Finally, the *allocation mechanism* allocates rewards based on reputation rankings. This in turn might result in an update in the local ledger. The model is dynamic and operates in *epochs*, with each epoch representing a discrete time step during which changes to the network structure occur, encompassing one complete iteration of the system's feedback loop. We will now discuss each mechanism in detail.

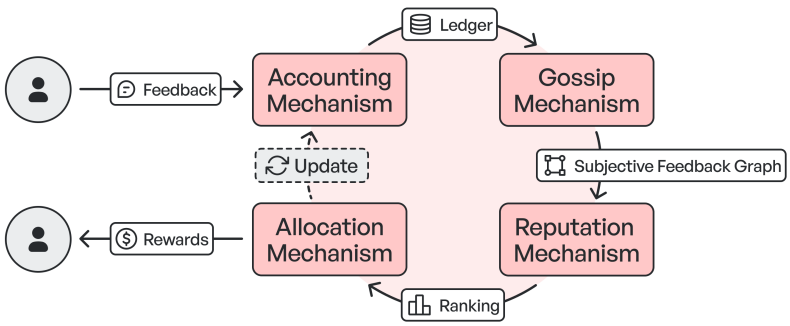


Figure 4.2: Merit-Based Tokenomics system model.

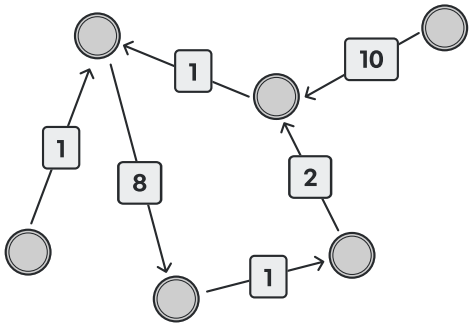


Figure 4.3: Example of a feedback graph. Edge weights represent the total feedback assigned by a participant about another participant.

4.3.1 Accounting Mechanism

We assume that peers might interact through various activities, such as making direct contributions, creating content, participating in discussions, or engaging in collaborative tasks. In response to these interactions, peers provide *feedback*, which can take the form of evaluations, ratings, or other reactions. This feedback is represented as a directed, weighted graph, denoted as $G = (V, E, w)$, and referred to as the *feedback graph*. In this graph, each node represents a peer in the network, while each directed edge indicates feedback provided from one peer to another. The weight of an edge, defined by the function $w : V \times V \to \mathbb{R}_{\geq 0}$, assigns a non-negative value that quantifies the cumulative feedback assigned.

An example of a feedback graph is shown in Figure 4.3, where the weights on the edges represent feedback quantified in arbitrary units. Our model is agnostic to the method of feedback generation, which can range from simple social reactions—such as likes, votes, or endorsements—to more complex algorithms that assign scores based on contributions. For instance, a "thumbs up" can be directly translated into a single unit, while systems like SourceCred [196] compute scores for actions such as code commits or issue resolutions based on predefined metrics. Similarly, feedback can come from explicit responses (e.g., ratings or endorsements) or from implicit signals (e.g., engagement levels or contribution frequency) as seen in platforms like Discourse [231] and GitHub [232]. In both cases, the resulting value is recorded in the personal ledger of the observing peer.

4.3.2 Gossip Mechanism

Peers exchange and propagate updates to the feedback graph using a peer-to-peer gossip protocol [233]. Through this mechanism, feedback data recorded in each peer's personal ledger is periodically shared across the network, allowing reputation data to evolve based on recent interactions. Each peer maintains its own *subjective* view of the global feedback graph, denoted as $G_i = (V_i, E_i, w_i)$, where V_i and E_i represent the known nodes and feedback links, and $w_i : V_i \times V_i \to \mathbb{R}_{\geq 0}$ assigns weights based on the latest received data.

Due to network delays or incomplete data propagation, the local graph G_i may differ from the views of other peers. When a peer receives new updates, such as the addition of a new edge or a change in the weight of an existing edge, it updates its local view and

records this information in its personal ledger. As peers continue to share more data, these subjective graphs gradually converge, resulting in a more consistent view of the network.

4.3.3 Reputation Mechanism

We assume that peers can discover the feedback graph through mechanisms such as a gossip protocol. The reputation data evolves with time and reflects the latest interactions by utilizing a peer-to-peer gossip protocol [233]. Specifically, each peer distributes the latest locally known weights in the feedback graph. Using this information, each peer i constructs its *subjective* feedback graph $G_i = (V_i, E_i, w_i)$. Every time a peer receives a new edge or a weight update from the network, it records this in its personal ledger and updates its subjective feedback graph G_i.

A reputation mechanism calculates and assigns reputation scores to every node in the subjective feedback graph. The reputation score reflects the level of contributions made by a node relative to others.

Definition 4.1 (Reputation Score) *A reputation score $R_i(G_i, j)$ is a non-negative value calculated and assigned to node j by the reputation mechanism of node i, given the subjective feedback graph G_i:*

$$R_i(G_i, j) \in \mathbb{R}_{\geq 0} \quad \forall j \in V_i \setminus \{i\}$$

4.3.4 Allocation Mechanism

Reputation scores are used as inputs for an allocation mechanism that determines how rewards are distributed among a set of nodes using an allocation policy. Specifically we assume some seed node i, which is also part of V. The seed node uses its local reputation scores to distribute rewards to other nodes according to their reputations at the end of each epoch.

Definition 4.2 (Allocation score) *An allocation score $A_i(G_i, j)$ is a non-negative value calculated and assigned to node j by the allocation mechanism of node i, based on the subjective feedback graph G_i and reputation scores $R_i(G_i, j)$:*

$$A_i(R_i(G_i, j), j) \in \mathbb{R}_{\geq 0} \quad \forall j \in V_i \setminus \{i\}$$

We do not make any specific assumptions about the allocation mechanisms but assume that nodes with higher reputation scores are more likely to receive rewards or receive larger rewards. The allocation of rewards depends on the specific seed node. If there are multiple seed nodes, a node will receive rewards separately from each seed node, and the total rewards will be the sum of these individual allocations.

One example of an allocation mechanism is *winner-takes-all*, where the node with the highest reputation receives all the rewards. Another popular example is the quadratic distribution [206], where rewards are distributed such that each node receives a portion of the total reward pool proportional to the square root of its reputation score. This ensures that while nodes with higher reputations receive more significant rewards, the rate of increase in rewards diminishes as reputations grow, preventing disproportionate advantages.

Different allocation mechanisms will have different applications and contexts where they are most effective. However, a detailed discussion on the specific properties and implications of various allocation policies is beyond the scope of this work.

4.4 Sybil Attack and Sybil Tolerance

In this section, we introduce the model for a Sybil attack on merit-based tokenomics, as presented in the previous section. The attacker executes the Sybil attack to inflate its reputation and subsequently gain disproportionately more rewards. We then discuss various strategies that attackers can employ to maximize the effectiveness of Sybil attacks. Finally, we present a model for Sybil tolerance, which quantifies how well a reputation mechanism can withstand a Sybil attack. We conclude by reporting on the most beneficial Sybil attack strategies against the most commonly used reputation mechanisms.

4.4.1 Sybil Attack

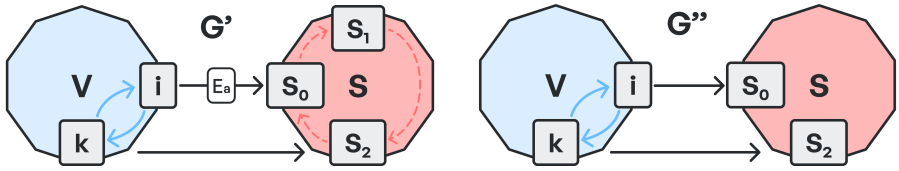
We model a Sybil attack as *strategic*, meaning the attacker first infiltrates into the system pretending to be an honest node, receiving legitimate feedback from other honest nodes, and then executes a Sybil attack. The attacker achieves this by creating fake identities (Sybil nodes) and fake edges connecting these identities to each other (Sybil edges). The weights of the Sybil edges can be arbitrary. We refer to the created subgraph consisting of Sybil nodes and edges as *Sybil region*. We assume the attacker knows the reputation and allocation mechanisms being used and can execute an optimal attack.

Definition 4.3 (Sybil Attack) *Given the feedback graph $G = (V, E, w)$, an attacker s_0 performs a Sybil attack σ_S by introducing the following elements to the feedback graph:*

- *A set of **Sybil nodes** $S = \{s_1, \ldots, s_m\} \cup \{s_0\}$, each of which is indistinguishable from an honest node by other nodes.*
- *A set of **Sybil edges** $E_S \subset S \times S$ with arbitrary edge weights $w_S : S \times S \to \mathbb{R}_{\geq 0}$.*
- *A set of **attack edges** $E_a \subset V \times S$ with weights $w_a : V \times S \to \mathbb{R}_{\geq 0}$.*

After an attack has been carried out, we obtain a modified feedback graph, denoted by $G' := (V \cup S, E \cup E_S \cup E_a, w \cup w_S \cup w_a)$. We denote by $G'' := (V \cup S, E \cup E_a, w \cup w_a)$ the modified feedback graph with Sybil edges removed.

Figure 4.4 illustrates an example of a Sybil attack. The attack has three Sybil nodes $\{s_0, s_1, s_2\}$. The attacker establishes Sybil edges between s_0, s_1, and s_2 with arbitrarily high weights, as these edges can be freely generated by the attacker without any constraints. In contrast, attack edges typically require the attacker to make some real contributions to receive feedback from honest reputable nodes. To infiltrate the network, the attacker using identities of s_0 and s_2 performs some contribution to receive feedback from some highly reputable nodes i and k in V, resulting in the creation of attack edges (i, s_0) and (k, s_2). As a result of this attack, the Sybil node s_1 receives positive reputation. The created cycle $(s_0, s_1), (s_1, s_2), (s_2, s_0)$ with high weights further increases the reputation of each Sybil node, thereby inflating their reputation scores and allowing them to disproportionately benefit during the allocation phase.



(a) The modified graph G' after the attack.

(b) The modified graph G'' with Sybil edges removed.

Figure 4.4: An example of a Sybil attack on graph $G = (V = \{k, i, ...\}, E = \{(k,i),(i,k),...\}, w)$ with Sybil nodes $S = \{s_0, s_1, s_2\}$, two attack edges $E_a = \{(i,s_0),(k,s_2)\}$ and Sybil edges $E_S = \{(s_0,s_1),(s1,s2),(s2,s_0)\}$.

4.4.2 Sybil Attack Strategies

Naive approaches to reputation systems that rely on basic global centrality measures, such as degree centrality, are highly susceptible to manipulation. These approaches assign importance based on the number of direct connections a node has, making it easy for an attacker to inflate their reputation simply by creating Sybil nodes and edges that connect them to the attacker node. This artificially increases the attacker node's degree centrality, thereby unfairly boosting its reputation.

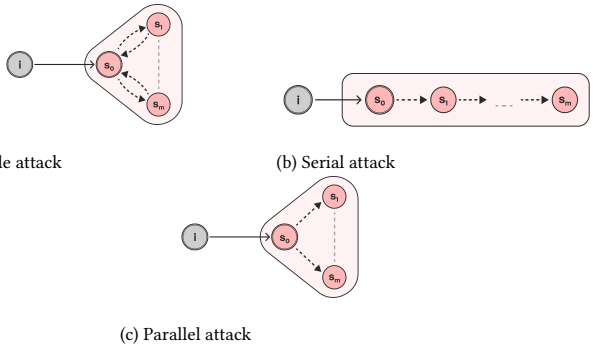
More complex global centrality measures, such as (weighted) PageRank, are also vulnerable to Sybil attacks [217]. An attacker can create a dense Sybil region, effectively increasing the PageRank of Sybil nodes, even if the Sybil nodes themselves have low individual ranks. Moreover, an attacker can perform successful Sybil attacks without the need to create any attack edges.

We consider more sophisticated attack that require the creation of attack edges. Typically, an attacker infiltrates the system by interacting with honest nodes, gradually collecting reputation through before executing a Sybil attack. Once sufficient reputation is gained, the Sybil attack is launched, and some inflated reputation is gained. To further amplify the effect, we also consider a *repeated Sybil attack*, where the attacker introduces a fresh batch of Sybil nodes in every epoch. This strategy compounds the attack's impact over time, as each new set of Sybil nodes reinforces the influence of the malicious node through an expanding web of attack edges.

Personalized reputation mechanisms [208] inherently limit the scope of Sybil attacks by tying reputation to connectivity with seed nodes. A node can only gain a positive reputation if it is connected to a seed node through some path or directly. In systems with multiple seed nodes the challenge for attackers increases significantly. To inflate the reputation of a malicious node in such systems, the attacker must establish paths to each relevant seed node. If no such paths are established, the malicious node's reputation score remains zero, effectively neutralizing the potential for Sybil attacks.

We now present three Sybil attack strategies, which, when combined, cover all Sybil attack strategies as shown in the previous works [211]. These strategies are illustrated in Figure 4.5.

Definition 4.4 (Sybil Attack Strategies) *An attacker node s_0 initially creates an attack edge (i, s_0) with some node $i \in V$ by making a contribution and receiving a feedback from node*



(a) Cycle attack

(b) Serial attack

(c) Parallel attack

Figure 4.5: Sybil attack strategies. A beneficial Sybil attack is a combination of these three strategies.

i. Subsequently, it creates m Sybil identities and Sybil edges, resulting in the modified feedback graph $G'(m)$. The Sybil edges are created in one of the following ways:

- ***Cycle attack**. The attacker node s_0 creates both incoming and outgoing edges with each other Sybil node, forming edges (s_0, s_k) and (s_k, s_0) for $k \in \{1, \ldots, m\}$.*
- ***Serial attack**. The attacker node s_0 creates a linear sequence of edges (s_0, s_1), and (s_k, s_{k+1}) for $k \in \{1, \ldots, m-1\}$.*
- ***Parallel attack**. The attacker node s_0 creates directed outgoing edges to each other Sybil node, forming edges (s_0, s_k) for $k \in \{1, \ldots, m\}$.*

We refer to the modified feedback graph $G\prime(m)$ as the graph obtained after a Sybil attack with exactly m Sybil nodes and the corresponding Sybil edges added according to one of the attack strategies defined above.

4.4.3 Sybil Tolerance

A *beneficial* Sybil attack occurs when an attacker successfully inflates the reputation of Sybil nodes, allowing them to receive a disproportionate share of rewards. The attacker is allowed to create an arbitrary number of Sybil identities and establish fake edges between them to boost their reputation within the Sybil network. However, any edge between honest nodes in V and Sybils in S must represent a real transaction. This means that the attacker must make genuine contributions to receive feedback from honest nodes, thereby creating legitimate directed edges connecting the honest network to the Sybil identities. To maximize their gains, the attacker might target highly reputable nodes to create these legitimate attack edges, as feedback from such nodes would significantly boost the reputation of the connected Sybil nodes.

We distinguish between two types of reputations in this context: *deserved* and *inflated*. The *deserved* reputation represents the value earned through legitimate feedback provided by honest nodes. In contrast, the *inflated* reputation includes both the deserved reputation and any additional reputation artificially increased through fake edges between Sybil iden-

tities. The goal of the attacker is to inflate the reputation as much as possible, maximizing the cumulative reputation of the Sybil nodes through these fake edges.

Definition 4.5a (Inflated Reputation) *Given an arbitrary seed node $i \in V$, the inflated reputation $\omega^+(\sigma_S)$ for a Sybil attack σ_S is the cumulative reputation score gained by the Sybil nodes in S calculated by i over the modified feedback graph G':*

$$\omega^+(\sigma_S) = \sum_{s \in S} R_i(G'_i, s)$$

Definition 4.5b (Deserved Reputation) *Given an arbitrary seed node $i \in V$, the deserved reputation $\omega(\sigma_S)$ for a Sybil attack σ_S is the cumulative reputation score gained by the Sybil nodes in S calculated by i over the modified feedback graph G'' with the Sybil edges removed:*

$$\omega(\sigma_S) = \sum_{s \in S} R_i(G''_i, s)$$

Definition 4.5c (Attacker's Gain) *The attacker's gain of a Sybil attack given $\omega(\sigma_S) > 0$ is defined as the ratio of the inflated and deserved reputation:*

$$Gain(\sigma_S) = \frac{\omega^+(\sigma_S)}{\omega(\sigma_S)}$$

Sybil tolerance is modeled as an upper bound on the attacker's gain for a Sybil attack, denoted as σ_S, on a feedback graph G. We formally define a Sybil-Tolerant reputation mechanism as follows:

Definition 4.6 (Sybil Tolerance) *A reputation mechanism is* Sybil-Tolerant *if, for every Sybil attack, the attacker's gain is bounded by some constant $c \geq 1$ for arbitrarily large sets S of Sybil nodes with any arrangement of Sybil edges, that is, if*

$$\lim_{|S| \to \infty} \frac{\omega^+(\sigma_S)}{\omega(\sigma_S)} \leq c$$

In theory, a Sybil-resistant mechanism is characterized by having $c = 1$, meaning the attacker cannot gain any inflated reputation through Sybil identities and can only earn what is deserved from feedback provided by honest nodes. However, in practice, achieving full Sybil resistance ($c = 1$) in open, decentralized systems is extremely challenging due to the ease of creating pseudonymous identities and the potential for Sybil nodes to receive legitimate feedback from honest participants. Therefore, while mechanisms like restricting nodes to self-assessments can theoretically achieve $c = 1$, they severely limit the utility of the reputation system by ignoring valuable indirect feedback and collaboration.A notable example of this type of restrictive mechanism is the use of direct reciprocity strategies, such as Tit-for-Tat [203]. In the Tit-for-Tat strategy, nodes reciprocate interactions, meaning they only provide positive feedback to nodes that have previously contributed to them in some way. This approach, widely known from its use in file-sharing protocols like BitTorrent [234], ensures that nodes cooperate only if they receive direct value from the interaction.

On the other hand, if the feedback from all peers is taken into account, Sybil tolerance is not achievable, as reported by Seuken and Parkes [221]. The attacker's gain from a Sybil attack can become unbounded by creating a large Sybil region and amplifying the weights of Sybil edges. In practice, this means that the attacker can effectively receive a substantial reward for each allocation.

4.4.4 Sybil-Tolerance of Existing Reputations

We evaluate three widely-used reputation mechanisms considered to be Sybil-resistant: personalized PageRank [235], personalized Hitting Time [208], and BarterCast MaxFlow [207]. In this section, we present examples demonstrating that these mechanisms are not Sybil-resistant and, in some cases, fail to even achieve Sybil-tolerance.

Personalized PageRank: The personalized PageRank algorithm (PPR) is a variant of the standard PageRank algorithm, tailored to measure the importance of nodes relative to a specific seed node i. The algorithm works as follows: A random walk is initiated from the seed node i. At each step, with probability α, the walk terminates and 'teleports' back to the seed node i, and with probability $1-\alpha$, the walk moves to a randomly selected neighboring node. The PPR reputation of a node j is the steady-state probability that the random walk initiated at i will be at j at any given step. In other words, it represents the probability that a random walker starting from the seed node i will be found at node j after any number of steps, assuming the process has reached equilibrium. In practice, this is estimated by running multiple random walks starting from the seed node i and counting the number of times node j is reached.

One significant issue with PageRank is that random walks can become trapped in the Sybil region, especially when a cycle attack is executed. In the global version of PageRank, this can occur even without the presence of attack edges. The most effective method for attacking PageRank is through a cycle attack, where the attacker creates edges with large weights among Sybil nodes. This manipulation increases the total number of encounters in a random walk, thereby enabling unbounded inflation of the reputation scores of the Sybil nodes and making PageRank not Sybil-tolerant.

PPR introduces a level of Sybil tolerance due to its damping factor α. This damping factor ensures that at each step, there is a probability α of returning to the seed node, which reduces the likelihood of random walks becoming trapped in the Sybil region. PPR was previously reported as Sybil-resistant, but this is true only in the absence of attack edges, where no random walks would reach any of the Sybil nodes. However, when attack edges are present, the attacker can still obtain a gain from a Sybil attack, especially when α is low.

Personalized Hitting Time: The Personalized Hitting Time (PHT) algorithm is a variation of the random walk-based algorithms. Unlike PPR, which focuses on steady-state probabilities and the long-term presence of nodes, PHT concentrates on the first encounter of nodes during the random walk. It measures the expected number of steps a random walker, starting from a seed node i, will take to reach target node j for the first time.

Compared to PageRank, PHT is less affected by cycle attacks. However, PHT remains susceptible to serial attacks, where Sybil identities are connected in a linear sequence. In this scenario, each Sybil node in the sequence gains some positive reputation as the random walk progresses through the chain, ultimately leading to the inflation of the reputations of

all Sybil nodes along the path. Similar to PPR, PHT is not inherently Sybil-resistant but can achieve some level of Sybil tolerance by using a high teleportation probability α.

MaxFlow: The MaxFlow (MFW) algorithm is a classical approach in network flow theory that determines the maximum possible flow from a source node to a target node in a flow network. Given a directed graph $G = (V, E, w)$ where each edge $(i, j) \in E$ has a capacity $w \geq 0$, the objective is to find the maximum flow from a source node i to a sink node j such that the flow on each edge does not exceed its capacity and the incoming flow equals the outgoing flow for every node except i and j.

MFW was proposed for use in reputation systems because it effectively mimics the idea of credit networks [230], where the flow of value or contributions can be tracked through paths in a network. In such systems, the reputation of a node is determined by its ability to contribute value that can reach the target node, considering all possible routes. This approach ensures that the amount a node can gain is bounded by its aggregated contributions, making it a robust measure of a node's overall influence and trustworthiness within the network.

Despite its theoretical benefits, MFW-based reputation mechanisms are vulnerable to Sybil attacks due to how they aggregate flow across multiple paths. In a parallel attack (shown in Figure 4.5c), an attacker creates multiple Sybil identities, each connected directly to the attacker's main node s_0. Since MFW sums the flow through all paths, each additional Sybil identity increases the total flow capacity from honest nodes to the attacker, allowing the attacker to artificially inflate its cumulative reputation. This results in an unbounded gain because there is no constraint on the number of Sybil identities an attacker can create.

4.4.5 Bounding Properties for Sybil-Tolerance

In this section we define three desirable properties of reputation mechanisms to achieve the Sybil-Tolerance.

The first property we define aims to bound the effectiveness of parallel and cycle Sybil attacks. These attacks are effective because reputation mechanisms typically treat each new edge as an independent source of feedback, resulting in an inflated reputation for the attacker. To address this issue, we define a property called the *Parallel Attack Bound*. This property ensures that, regardless of the number of Sybil nodes introduced, the cumulative reputation gain remains bounded, so the attacker's inflated reputation does not exceed what would be achieved by a single Sybil node.

Definition 4.7 (Parallel Attack Bound) *A reputation mechanism is parallel attack bound if, for any seed node $i \in V$ and for the set of Sybil nodes $s_1, s_2, ..., s_m$ generated through a parallel or cycle attack, the total reputation satisfies:*

$$\sum_{l=1}^{m} R_i(G_i'(m), s_l) \leq R_i(G_i'(1), s_1).$$

The second property we define is aimed at bounding the effectiveness of serial Sybil attacks. In such attacks, the attacker extends the path of Sybil nodes, with each new Sybil contributing additional feedback along the chain, thereby inflating the attacker's reputation. To address this, we introduce the *Serial Attack Bound*. This property ensures that the inflated reputation of a serial attack is bounded, even as the length of the Sybil paths

increases, preventing the attacker from achieving unbounded gains by simply extending the chain of Sybil nodes.

Definition 4.8 (Serial Attack Bound) *A reputation mechanism is serial attack bound if, for any seed node $i \in V$ and for the set of Sybil nodes $s_1, s_2, ..., s_m$ generated through a serial attack, the total reputation satisfies:*

$$\lim_{m \to \infty} \sum_{l=1}^{m} R_i(G_i'(m), s_l) < \infty.$$

The *Bounded Transitivity* property ensures that reputation values cannot be inflated through paths that include low-reputation nodes. Our proposed decay mechanisms, particularly transitivity decay, directly enforce this property by reducing the influence of nodes as their distance from the seed node increases. This means that reputation cannot be disproportionately amplified through long chains of intermediaries, which often include low-reputation or Sybil nodes. By integrating decay mechanisms, we ensure that the reputation system adheres to Bounded Transitivity, enhancing its resistance to manipulation. This principle has been explored in reputation algorithms that incorporate MaxFlow concepts, such as BarterCast [207].

Definition 4.9 (Bounded Transitivity) *A reputation mechanism satisfies* Bounded Transitivity *if, for any seed node $i \in V$ and any node $j \in V_i$ such that there are N node-disjoint, simple paths P_d (where $d = 1, 2, \ldots, N$) from i to j, it holds that:*

$$R_i(G_i, j) \leq \sum_{d=1}^{N} \min\{R_i(G_i, k) \mid k \in P_d\}.$$

4.5 MeritRank: Sybil Tolerant Reputations

In this section, we introduce MeritRank, a set of techniques designed to enhance the Sybil tolerance of existing reputation mechanisms.

4.5.1 Bounding Properties in Practice

To achieve the bounding properties discussed in the Section 4.4.5 in practice, we propose four generic techniques for existing reputation mechanisms. Specifically, we introduce relative feedback, decay on transitivity, and decay on connectivity as heuristics to achieve bounds on Sybil attacks. Additionally, we consider epoch decay due to its popularity in existing reputation mechanisms.

Relative Feedback. In a Sybil attack, the attacker can control the edge weights of Sybil nodes, potentially assigning arbitrarily large values to inflate the influence of these connections. Without proper normalization, these artificially high weights can distort the reputation mechanism, as the system might rely excessively on the absolute values of the incoming edges. To counteract this, Relative Feedback introduces a normalization step to each edge weight, ensuring that no single node can amplify its reputation through artificially inflated connections.

We introduce the Normalized Graph G^N of a weighted graph $G = (V, E, w)$ in the following way: for every node $k \in V$, let $\mathcal{N}(k)$ be the set of its neighboring nodes in G. The corresponding *Normalized Graph* $G^N = (V, E, w^N)$ is defined such that:

$$w^N(k, j) = \frac{w(k, j)}{\sum_{l \in \mathcal{N}(k)} w(k, l)}, \quad \forall (k, j) \in E.$$

The normalization process is applied to every subjective graph, adjusting all edge weights accordingly. For flow-based algorithms, this normalization is applied prior to calculating the reputation scores. For random-walk based mechanisms, the normalization is incorporated dynamically during the calculation of the transition probabilities, with each step reflecting the probability of moving from node k to node j based on $w^N(k, j)$.

This mechanism is the key to achieve Bounded Transitivity property and contributes to the Parallel Attack Bound. It ensures that the reputation is always considered in the context of its neighbors. Attackers cannot inflate reputations by assigning large weights to edges connected to Sybil nodes, as the normalization limits their impact.

Transitivity Decay. The purpose of this decay is to decrease the reputation of nodes as their distance from the seed node increases. This discourages the creation of long chains, such as those used in serial attacks, by ensuring that nodes further from the seed receive progressively lower scores.

In random-walk-based mechanisms, this concept is naturally implemented using a teleportation probability α. With a probability of α, a random walk returns to the seed node, limiting the influence of nodes further down the path. Typically, α is set between 0.1 and 0.2 [236]. While we reuse this idea, we reinterpret α not merely as a teleportation probability but as a decay factor that systematically reduces a node's reputation contribution based on its distance from the seed node. Instead of merely terminating random walks, α serves to attenuate the influence of nodes further away, ensuring that the reputation impact diminishes with each additional hop in the network.

In a flow-based reputation system, transitivity decay is implemented by introducing a reduction factor applied to the flow. Specifically, the flow w from node i to node j is modified to $w'(e) = (1 - \alpha)^d \times w(e)$, where d is the distance from the seed node to node j.

Transitivity Decay directly enforces the Serial Attack Bound by diminishing the influence of nodes further from the seed node, thus limiting the cumulative reputation that can be gained through serial attacks.

Connectivity Decay. Sybil nodes often form dense clusters that connect to the rest of the network through a small number of intermediary nodes known as *cut-vertices*. A *cut-vertex* is a node whose removal increases the number of connected components in a graph, potentially isolating certain nodes. Identifying and accounting for cut-vertices is essential in mitigating the influence of nodes that depend on such vulnerable connections.

The *connectivity decay* mechanism adjusts the reputation scores of nodes that depend on cut-vertices, thereby reducing the influence of potential Sybil clusters. The adjusted reputation score with connectivity decay for a node $j \in V_i$ with respect to a seed node i, is defined as:

$$R_{i,\beta}(G_i, j) = \begin{cases} (1-\beta) \cdot R_i(G_i, j), & \text{if } I(i,j) = 1, \\ R_i(G_i, j), & \text{otherwise,} \end{cases}$$

where $0 \leq \beta \leq 1$ is the decay factor, and $I(i, j)$ is an indicator function that determines whether the connection between nodes i and j is considered vulnerable due to low *local node connectivity*.

The concept of *local node connectivity* between two nodes i and j, denoted $\kappa(i, j)$, is defined as the minimum number of nodes (excluding i and j) whose removal would disconnect i from j. Equivalently, $\kappa(i, j)$ is the maximum number of *node-independent* paths between i and j, where node-independent paths are paths that do not share any nodes other than the endpoints. We introduce a *connectivity threshold* t (with $t \geq 1$) to determine the acceptable level of connectivity between nodes. If the local node connectivity $\kappa(i, j)$ is less than or equal to t, the connection between i and j is considered vulnerable and subject to decay. The indicator function $I(i, j)$ is then defined as:

$$I(i,j) = \begin{cases} 1, & \text{if } \kappa(i,j) \leq t, \\ 0, & \text{otherwise.} \end{cases}$$

In practice, calculating $\kappa(i, j)$ directly can be computationally intensive for large networks. To address this, we perform a large number of random walks initiated from a seed node, with each node calculating locally based on its own subjective feedback graph. The indicator function $I(i, j)$ is estimated by analyzing the frequency of random walks from node i to node j that pass through intermediary nodes. This estimation is practical because random walks are computationally efficient and can be performed locally by every node using its subjective view of the network.

Let T_{ij} denote the total number of random walks from node i to node j, and let $T_{ij}(k)$ represent the number of these walks that pass through an intermediary node $k \in V \setminus \{i, j\}$. A node k is considered critical if the proportion of walks passing through k exceeds a threshold of $1/t$. Thus, the indicator function $I(i, j)$ is equivalently defined as:

$$I(i,j) = \begin{cases} 1, & \text{if } \max\limits_{k \in V \setminus \{i,j\}} \left(\dfrac{T_{ij}(k)}{T_{ij}} \right) \geq \dfrac{1}{t}, \\ 0, & \text{otherwise.} \end{cases}$$

This approximation relies on the idea that when $\kappa(i, j)$ is low, there are fewer node-independent paths between i and j, causing a higher proportion of random walks to pass through certain critical nodes. The threshold t determines the sensitivity of the connectivity decay mechanism. The higher t, the more node-independent paths are required to avoid the decay.

Connectivity decay penalizes nodes that are connected to the seed node through a small number of independent paths. In a parallel attack, an attacker creates multiple Sybil nodes directly connected to itself, which is then connected to the seed node through limited attack edges. By applying connectivity decay, the reputation influence of each Sybil node is reduced because they all rely on the same limited connections to the seed node, enforcing the Parallel Attack Bound. Similarly, connectivity decay impacts serial attacks by reducing

the influence of Sybil nodes that are connected to the seed node through a single, elongated path, thereby achieving the Serial Attack Bound.

Epoch Decay. Attackers might attempt to exploit the system by maintaining high reputations for their Sybil nodes based on the older feedback. To counter this, we introduce the concept of *epoch decay*, which reduces the influence of feedback as it ages. The epoch decay aims to require continuous positive contributions to maintain a high reputation. However, epoch decay might have unintended consequences. For instance, it could uniformly reduce the reputations of all participants and, as a result, potentially favor attackers who can rapidly generate new Sybil identities.

Given an epoch decay coefficient $0 \leq \gamma \leq 1$, we detail two methods for implementing this decay:

1. **Decay on Reputation Values**: This method dynamically adjusts reputation values over time to give more weight to recent contributions. The reputation update formula for a node j from the perspective of a seed node i at epoch $\tau + 1$ is:

$$R_i(G_i^{(\tau+1)}, j) = (1-\gamma) \cdot R_i(G_i^{(\tau)}, j) + R_i(G_i^{\Delta(\tau+1,\tau)}, j)$$

   Where, $G_i^\tau = (V_i^\tau, E_i^\tau, w_i^\tau)$ is the subjective feedback graph of node i at epoch τ, and $G_i^{(\Delta(\tau+1,\tau))} = (V_i^{(\tau+1)}, \Delta E_i^{(\tau+1)}, \Delta w_i^{(\tau+1)})$ represents the changes to the graph G_i^τ in epoch $\tau + 1$, which account for newly added edges and the weight updates for existing edges.

2. **Decay on Graph Weights**: This method applies decay directly to the weights of the graph's edges, progressively decreasing the impact of older edges. The weight update formula for an edge (i, j) at epoch $\tau + 1$ is:

$$w_\gamma^{(\tau+1)}(i,j) = \max\left(0, w^{(\tau+1)}(i,j) - (1-\gamma) \cdot w^{(\tau)}(i,j)\right)$$

4.6 Experiments

In this section, we present the results of a quantitative study on the MeritRank algorithm. Our evaluation focuses on the impact of Sybil attacks on reputation scores and examines how the proposed decay mechanisms influence these scores. Additionally, we assess the effect of decay mechanisms on the informativeness of the reputation system, i.e., the system's ability to still accurately rank honest nodes.

4.6.1 Experimental Set-Up

For our study, we consider the MakerDAO forum, one of the largest decentralized autonomous organizations (DAOs) to date [67]. MakerDAO is a decentralized platform built on the Ethereum blockchain that facilitates the issuance and management of the *DAI* stablecoin. The *DAI* token is an algorithmically stabilized cryptocurrency pegged to the US dollar. MakerDAO is governed by holders of the *MKR* utility token, which serves as the platform's governance token. *MKR* token holders participate in decision-making processes by creating and voting on proposals within the MakerDAO forum [237]. These proposals influence MakerDAO's operations, including adjustments to the parameters that maintain the stability of *DAI*.

We prepare a dataset for our experiments by parsing the forum activity using the provided API[1]. Our dataset encompasses all user interactions within the MakerDAO forum, including replies, likes, posts, and votes on proposals, spanning from June 24, 2019, to May 26, 2022 (153 weeks). Each action within the forum is quantified in terms of *work units*, following the framework defined by the SourceCred MakerDAO project[2]. For example, a post is evaluated based on the number of likes it receives, with each like contributing four work units to the post's total value. Using these work units, we construct a work graph where nodes represent users and entities (such as posts), and edges represent actions connecting them. For instance, if a user creates a post and another user likes it, this interaction is represented in the graph as a sequence of edges: from the user to the post (creation) and from the post to the user who liked it (feedback).

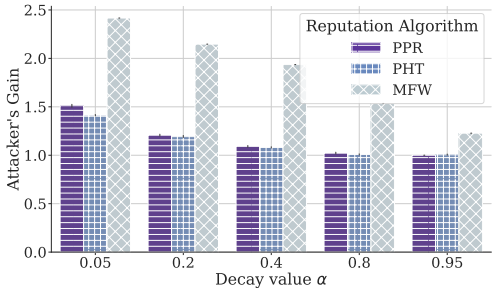
We then derive a feedback graph by compressing the work graph. In this process, intermediary entities like posts and their associated edges are first aggregated into direct edges between users, which are then combined into a single edge. The result is a simplified graph where the edge weight $w(i, j)$ between two users i and j reflects the cumulative relative feedback given from user i to user j. This edge weight is calculated by summing the work units of all interactions—such as likes, replies, and other forms of feedback—that user i has provided to the posts and comments made by user j.

We consider updates to the feedback graph over one-week epochs. Every epoch, the feedback graph is updated based on all forum interactions recorded that week. These updates appear as either new edges added between users or updated weights for existing edges. In total there are 153 epochs with peak activity occurring in week 149, when 1,528 new edges were added. By the end of epoch 153, the feedback graph comprised 2,057 nodes and 35,853 edges.

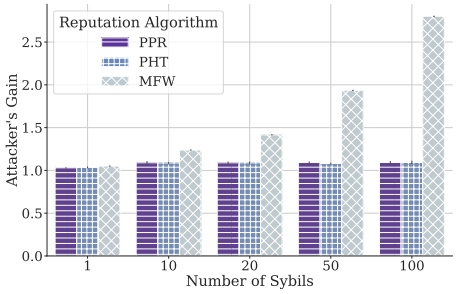
We create a simulation according to the model described in Section 4.3. For consistency, we maintain a fixed seed node outside the graph. As the feedback graph evolves with new edges and updated weights, this seed node remains continuously connected to the top 10 currently most reputable nodes with equal weights. To evaluate MeritRank, we implement two types of Sybil attacks: a *single Sybil attack* and a *repeated Sybil attack*.

Before initiating these attacks, we allow the system a 20-epoch grace period to establish initial reputations. At the end of this period, we designate one of the top 10 most reputable nodes as the attacker node. For the single Sybil attack, we simulate a scenario where an attacker infiltrates the system, executes a single attack with a varying number of Sybil nodes, and then withdraws, taking the profit.

In the repeated Sybil attack, the attacker consistently adds a new Sybil node every epoch to the system after the grace period. Each new Sybil node added is a new node with no prior reputation. The attacker node is selected once and fixed for the whole experiment run. This setup models a scenario where the attacker gradually increases its influence to poison the system and achieve larger cumulative gains. While less common in practice, this approach tests the system's resilience under worst-case conditions.

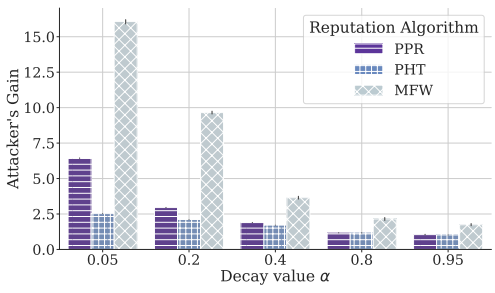


(a) Attacker's gain with 50 Sybil nodes.

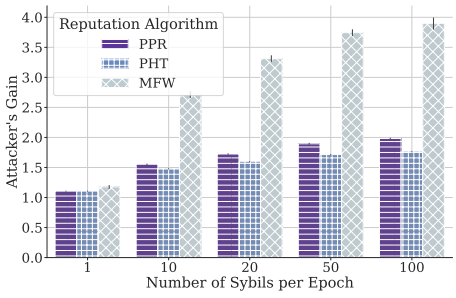


(b) Attacker's gain given a transitivity decay value α of 0.4

Figure 4.6: The attacker's gain from a single Sybil attack for the three reputation algorithms versus (a) the transitivity decay (α) and (b) the number of Sybil nodes.



(a) Attacker's gain with 50 Sybil nodes added per epoch.



(b) Attacker's gain given transitivity decay value α of 0.4

Figure 4.7: The attacker's gain from a repeated Sybil attack for the three reputation algorithms versus (a) the transitivity decay (α) and (b) the number of Sybil nodes per epoch.

4.6.2 Transitivity Decay

We implement the Personalized PageRank (PPR), Personalized Hitting Time (PHT), and MaxFlow (MFW) algorithms, incorporating transitivity decay to assess the robustness of these reputation mechanisms against Sybil attacks. For each reputation mechanism, we introduce a varying number of Sybil nodes and corresponding Sybil edges, following the most effective attack strategy identified for each reputation in Section 4.4.4. For all reported figures involving varying parameters, we repeat the simulation 10 times and present the average along with the standard error. Note that the standard error is not visible in the figures as its value is negligibly small.

We report the attacker's gain from a single Sybil attack executed right after the grace period in Figure 4.6. In Figure 4.6a, we illustrate the effect of transitivity decay as a function of the decay value α. We vary the decay value α from 0.05, representing minimal transitivity decay, to 0.95, where the most distant edges are severely decayed. Setting $\alpha = 0$ is not possible, as it would prevent random walks from terminating. We set the number of Sybil nodes to 50. The results show that the attacker can gain up to 1.5 to 2.5 times when there is minimal transitivity decay. However, as the decay value increases, the attacker's gain decreases.

Figure 4.6b further examines the effect of increasing the number of Sybil nodes on the attacker's gain. Even as the number of Sybil nodes increases, the attacker's gain does not grow significantly except for MFW. Notably, the MFW mechanism shows a higher attacker's gain compared to PPR and PHT.

To further analyze the cumulative impact of Sybil attacks over time, we execute repeated Sybil attacks and report the attacker's gain at the end of 153 epochs. Our results are presented in Figure 4.7. The repeated Sybil attack clearly shows that the attacker can gain more compared to the single Sybil attack, which is especially evident for MFW.

Our findings indicate that transitivity decay can successfully decrease the attacker's gain, especially with higher decay values. Among the reputation mechanisms tested, PHT results in the lowest attacker's gain for both single and repeated Sybil attacks. MFW has poor Sybil tolerance because it aggregates trust additively across all available paths, allowing attackers to inflate their reputation despite the decay. Moreover, MFW is computationally more intensive than PPR and PHT, making it less desirable in practice.

PPR performs worse than PHT because it is more vulnerable to cycle attacks. In PPR, reputation is distributed based on the stationary distribution of random walks with restarts, which can be influenced by the presence of cycles or loops in the network. Attackers can exploit this by creating cyclic structures among Sybil nodes to trap the random walks and thereby absorb more reputation. This results in higher reputation scores for Sybil nodes in PPR compared to PHT.

The intuition behind transitivity decay is that it is spatial—the further a node is from the seed node in terms of network distance, the more its reputation is decayed. Consequently, nodes that are farther away receive exponentially less reputation than those closer to the seed node. This mechanism is particularly effective against serial attacks, where an attacker attempts to build a long chain of Sybil nodes to reach the seed nodes. As a result, transitivity decay directly addresses PHT's vulnerability to serial attacks.

[1] `https://forum.sky.money`
[2] `http://makerdao.sourcecred.io`

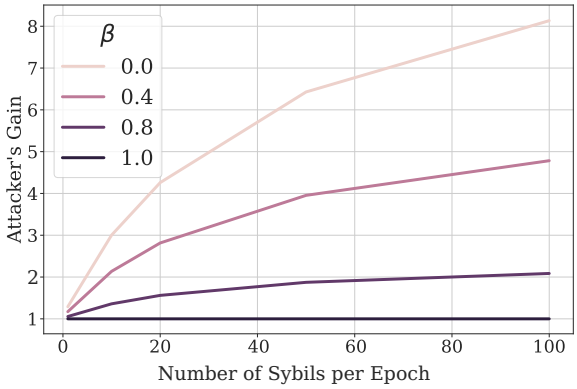


Figure 4.8: The attacker's gain versus the number of Sybils added per epoch given the connectivity decay β with the reputation algorithm PHT. The connectivity decay is fixed at $\alpha = 0.05$.

Due to its superior performance in mitigating Sybil attacks, we use PHT as the base algorithm for all later experiments. For PHT, we select the serial attack as the most effective one observed in prior tests. For the subsequent experiments, we only report the repeated Sybil attack, as it has more pronounced effects on the network's reputation system.

4.6.3 Connectivity Decay

For the connectivity decay experiments, we set the transitivity decay coefficient to a small value of $\alpha = 0.05$ and the connectivity threshold to $t = 1$. This configuration targets nodes that are connected to the graph exclusively through a single intermediary node, a typical characteristic of Sybil nodes. This decay is also spatial in its nature, like the transitivity decay.

Figure 4.8 illustrates the impact of connectivity decay on the attacker's gain. When there is no connectivity decay ($\beta = 0$), the attacker's gain increases sub-linearly with the number of Sybil nodes. However, as the connectivity decay coefficient β increases, the attacker's gain decreases significantly. Notably, with $\beta = 1.0$, the attacker's gain is completely eliminated because the reputation scores of Sybil nodes connected solely through a single intermediary drop to zero.

In Figure 4.9, we show that combining connectivity and transitivity decay mechanisms results in greater Sybil tolerance compared to applying each mechanism individually. Specifically, when both the transitivity and connectivity decay coefficients are set to 0.3, the attacker's gain remains limited to 1.5.

Our experiments demonstrate that MeritRank effectively mitigates the attacker's gain through spatial decay mechanisms, even against a significant repeated Sybil attack. Transitivity decay penalizes nodes that are too distant from the seed node, while connectivity decay penalizes nodes that are sparsely connected. Consequently, transitivity decay reduce s the effectiveness of serial attacks, while connectivity decay addresses parallel and cycle attacks. By combining both mechanisms, MeritRank achieves Sybil tolerance as defined

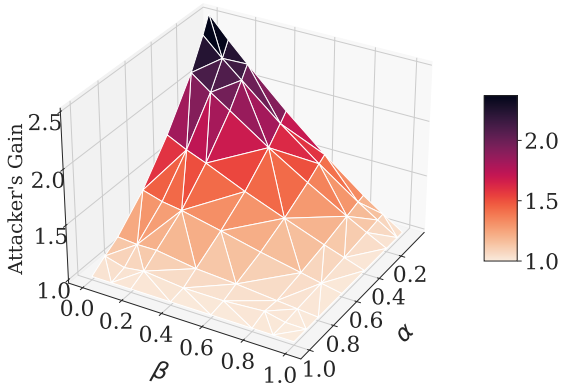


Figure 4.9: The joint effect of the transitivity decay (α) and connectivity decay (β) mechanisms on the attacker's gain given 50 Sybil nodes added per epoch with the reputation algorithm PHT.

in Definition 4.6, with a threshold of $1 \leq c \leq 2$.

4.6.4 Epoch Decay

We conduct experiments to evaluate the effectiveness of epoch decay in mitigating Sybil attacks by applying it to both reputation values and edge weights. The experiments simulate a repeated Sybil attack scenario with 50 Sybil nodes introduced per epoch, while the transitivity decay coefficient is fixed at $\alpha = 0.05$. The results, presented in Figure 4.10, show the attacker's gain across different epoch decay values. The findings indicate that epoch decay has minimal impact on the attacker's gain, regardless of whether it is applied to reputation values or graph weights. In both cases, the attacker's gain remains consistently high, with only minor fluctuations as the decay value increases.

A notable insight from our experiments is that epoch decay fails to address repeated Sybil attacks effectively. The underlying intuition behind epoch decay is to limit the cumulative influence of nodes over time by penalizing their contribution across epochs. This approach is intended to counter specific attack strategies where an attacker invests in creating a few attack edges and repeatedly exploits them by generating additional Sybil regions. However, our results demonstrate that this mechanism can be easily circumvented by maintaining at least one attack edge and introducing new Sybil nodes each epoch. As a result, the cumulative gain from the attack remains unaffected, rendering epoch decay ineffective in mitigating repeated Sybil attacks.

4.6.5 Effect of Decays on Informativeness

In the previous experiments, we demonstrated how decay mechanisms can effectively limit Sybil attacks. However, these mechanisms are applied universally, even in the ab-

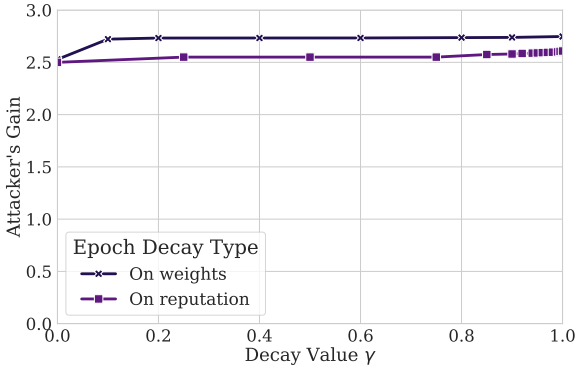


Figure 4.10: The attacker's gain versus epoch decay (γ) with 50 Sybil nodes added per epoch with reputation algorithm PHT.

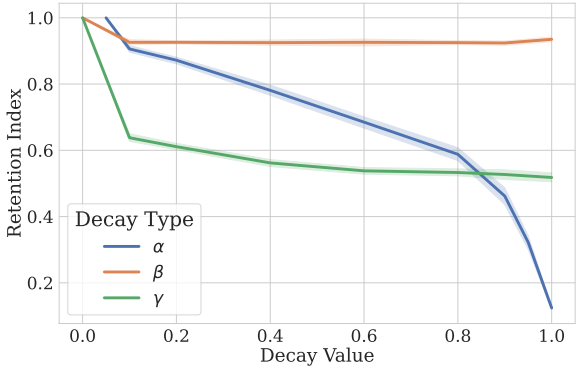


Figure 4.11: The Retention Index for the top 100 nodes given a reputation algorithm PHT versus the decay values for the decay mechanisms: transitivity (α), connectivity (β) and epoch (γ).

sence of Sybil nodes, which can unintentionally degrade the reputation scores of honest participants. Specifically, transitivity decay favors participants close to highly reputable nodes, connectivity decay prioritizes well-connected nodes, and epoch decay penalizes participants who are not consistently active. To quantify this unintended impact, we introduce a metric called *informativeness*, which evaluates how well the original reputation scores of honest participants are preserved after applying decay. To measure informativeness, we execute our simulation applying decay mechanisms without executing any Sybil attacks, reporting the informativeness at the end of epoch 153. We fix $t = 1$ for connectivity decay.

We use two complementary metrics to measure informativeness: the Retention Index for the top N nodes and the Mean Proportional Deviation (MPD). The Retention Index measures how many of the original top N nodes remain in the top N ranking after applying decay. This is particularly useful in scenarios where maintaining the composition of top contributors is important, for example, an allocation score that distributes all rewards to the top 100 most reputable nodes. The MPD, on the other hand, measures the exact deviations in reputation values. This metric is valuable in situations where rewards are allocated proportionally to reputation scores.

The Retention Index quantifies the preservation as the intersection of set of top N nodes before and after applying decay. This metric ranges from 0 (no overlap between the sets) to 1 (complete overlap). Let N denote the number of top nodes considered, and let $d \in [0,1]$ represent the value of the decay parameter for a given decay mechanism. We define $V_N^d \subseteq V$ as the set of the top N nodes with the highest reputation scores after applying decay, and $V_N^0 \subseteq V$ as the corresponding set before applying decay. The Retention Index, $R(d)$, is defined as:

$$R(d) = \frac{|V_N^0 \cap V_N^d|}{|V_N^0|}$$

Figure 4.11 illustrates the Retention Index for the top 100 nodes versus the decay value for all three decay mechanisms. The results indicate that connectivity decay (β) maintains the highest overlap with the baseline ranking across all decay values. Transitivity decay (α) exhibits a steady linear decline in overlap as decay increases, with higher decay values leading to a significant divergence from the baseline ranking. Epoch decay (γ) shows a significant drop at a decay value of 0.1. With this value 63 % of the nodes retain their rankings in the top 100. The index continues to decrease with further decay but stabilizes, maintaining approximately 50 % overlap when $\gamma = 1.0$.

The decline in the Retention Index with increasing decay is a direct result of the mechanisms prioritizing nodes closer to the seed, those with stronger connectivity, or participants with consistent activity. Our results indicate that informativeness largely depends on the connectivity and activity levels of participants. In the case of MakerDAO, the top 100 nodes are densely connected and highly active, with approximately 60% participating in at least half of the epochs. This suggests that the decay mechanisms have minimal impact on informativeness for the most central and consistently active participants, while peripheral or less active nodes may experience greater penalties.

Our second metric, MPD, quantifies the deviation of reputation scores from their original values when decay mechanisms are applied. For a given node $i \in V$, the proportional deviation of the reputation score R_i under a decay parameter d is defined as follows:

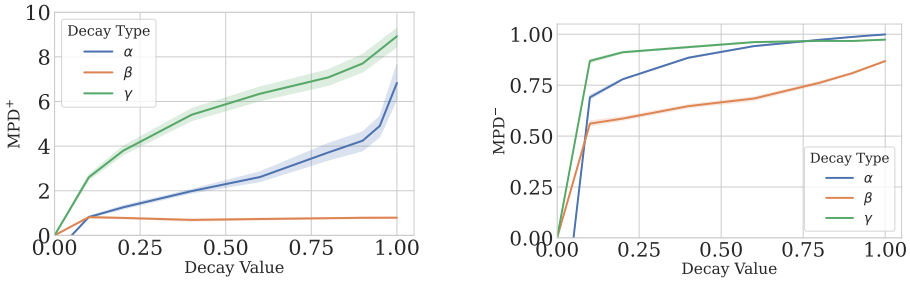


Figure 4.12: The positive and negative mean proportional deviation (MPD^+ and MPD^-) versus the decay value for the three decay mechanisms with the reputation algorithm PHT.

$$\delta_i(d,k) = \frac{R_{i,decay=d}(G_i,k) - R_{i,decay=0}(G_i,k)}{R_{i,decay=0}(G_i,k)}$$

The MPD is then separated into its positive and negative components. $MPD^+(d)$ captures the positive deviations (increases in reputation) and $MPD^-(d)$ captures the absolute values of the negative deviations (decreases in reputation):

$$MPD^+(d) = \frac{1}{|\{k \in V \setminus \{i\} \mid \delta_i(d,k) > 0\}|} \sum_{\substack{k \in V \setminus \{i\} \\ \delta_i(d,k) > 0}} \delta_i(d,k),$$

$$MPD^-(d) = \frac{1}{|\{k \in V \setminus \{i\} \mid \delta_i(d,k) < 0\}|} \sum_{\substack{k \in V \setminus \{i\} \\ \delta_i(d,k) < 0}} |\delta_i(d,k)|.$$

Figure 4.12 illustrates the behavior of $MPD^+(d)$ and $MPD^-(d)$ under the three decay mechanisms, averaged across 10 independent runs. The results demonstrate a clear divergence in how the different decay mechanisms influence the informativeness of the reputation scores.

For positive deviations (MPD^+), both transitivity decay and epoch decay exhibit a consistent and pronounced increase as decay values rise. In the case of transitivity decay, as the decay value increases, nodes closer to the seed node accrue disproportionately high reputation scores relative to their baseline. At the maximum decay value ($\alpha = 1.0$), direct neighbors of the seed node experience an average increase in their reputation scores by a factor of 7. A similar trend is observed for epoch decay, resulting in high deviations even with smaller decay values.

In contrast, connectivity decay shows a relatively stable and flat trend after an initial increase. This stability arises because connectivity decay selectively penalizes nodes with low local connectivity (i.e., nodes that are reachable through only a single path given t is 1). In the MakerDAO graph 23 % of the nodes have 0 or 1 incoming connections, which would be target for the penalty.

For negative deviations (MPD^-), all three decay mechanisms exhibit a rapid initial increase, followed by slower, steady growth as the decay value rises. At $\alpha = 1.0$, transitivity

decay leads to the complete nullification of all reputation scores, except for those of the direct neighbors of the seed node, leading to $\delta_i(1.0, k) = -1$. A similar effect is observed with epoch decay, where older contributions are systematically diminished, resulting in a substantial degradation of all reputation scores. In contrast, the increase in MPD^- for connectivity decay is less pronounced, this is because it does not uniformly reduce reputation scores across the graph. Specifically, $\delta_i = -1$ for the nodes that are sparsely connected, and $\delta_i = 0$ for the others.

Our experiments reveal that connectivity decay preserves informativeness better than transitivity and epoch decay for both the Retention Index and MPD metrics. This outcome is expected, as the connectivity decay affects a smaller subset of nodes within the graph.

4.7 Conclusion

The advances in complex blockchain applications have exposed the limitations of naive tokenomics, which often rely on simple models of monetary incentives for token holders. Merit-based reputation schemes that reward active contributors represent a promising direction for the development of novel blockchain applications such as DAOs. However, the application of reputation in decentralized environments is constrained by the *reputation trilemma*, which posits that a system cannot simultaneously be generalizable, trustless, and Sybil-resistant. We argue that feedback aggregation mechanisms offer the best approach to overcome this challenge as they maintain the trustless property essential to decentralization and are generalizable across various contexts. However, these mechanisms are inherently vulnerable to Sybil attacks, where malicious actors can inflate their reputation scores by creating multiple fake identities. To address this issue, we propose MeritRank, Sybil-tolerant reputations based on feedback aggregation with three decay mechanisms—transitivity decay, connectivity decay, and epoch decay.

Using a dataset of MakerDAO participant interactions, we experimentally demonstrate that MeritRank effectively limits the gains of attackers employing Sybil strategies. Furthermore, we investigate informativeness, i.e. the ability to accurately preserve the reputation of honest participants. Our experiments indicate that a combination of transitivity decay and connectivity decay achieves a desirable level of Sybil tolerance while preserving higher informativeness. In contrast, our findings reveal that epoch decay does not enhance Sybil tolerance and can even be counterproductive, as attackers can exploit it by continuously introducing new Sybil identities.

In conclusion, MeritRank offers a practical and effective solution for addressing the reputation trilemma. It allows for the application of different decay parameter choices to balance the trade-offs between informativeness and Sybil tolerance. Future work could explore alternative heuristics to optimize this balance and further investigate the counterproductive effects of certain mechanisms, such as epoch decay, to mitigate unintended vulnerabilities in reputation systems.

5

Sustainable Cooperation in Peer-To-Peer Networks

Traditionally, peer-to-peer systems have relied on altruism and reciprocity. Although incentive-based models have gained prominence in new-generation peer-to-peer systems, it is essential to recognize the continued importance of cooperative principles in achieving performance, fairness, and correctness. The lack of this acknowledgment has paved the way for selfish peers to gain unfair advantages in these systems. As such, we address the challenge of selfish peers by devising a mechanism to reward sustained cooperation.

Instead of relying on global accountability mechanisms, we propose a protocol that naturally aggregates local evaluations of cooperation. Traditional mechanisms are often vulnerable to Sybil and misreporting attacks. However, our approach overcomes these issues by limiting the benefits selfish peers can gain without incurring any cost. The viability of our algorithm is proven with a deployment to 27,259 Internet users and a realistic simulation of a blockchain gossip protocol. We show that our protocol sustains cooperation even in the presence of a majority of selfish peers while incurring only negligible overhead.

5.1 Introduction

Despite a wealth of experimentation with incentives in peer-to-peer networks, even such well-developed projects as Bitcoin and BitTorrent [234] do not directly address the fundamental problem of cooperation in these systems. The profit-seeking incentive for Bitcoin miners on its own does not provide guarantees for the sharing of historical blocks, which is accomplished on an altruistic basis [238]. BitTorrent, as well, critically relies on pure altruism for content seeding, after the tit-for-tat phase [239].

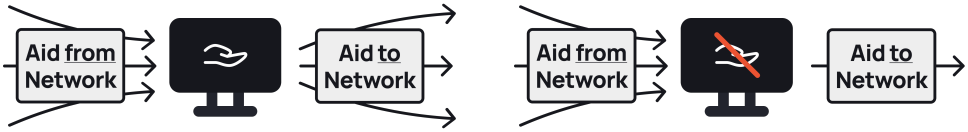
As such, selfish nodes pose a significant risk to the overall health and functioning of peer-to-peer networks. More specifically, the Bitcoin blockchain is vulnerable to a type of attack known as *selfish mining* [39]. In this deceptive mining strategy, nodes withhold successfully mined blocks to create a fork and get ahead of the longest public chain. This leads to two issues: first, it creates a scenario in which dishonest nodes have an unfair advantage and receive more mining rewards, and second, it degrades network performance [240]. Both of these problems have detrimental effects. The former issue results in a situation where honest nodes either stop participating altogether due to a lower probability of receiving block rewards or adopt the selfish mining strategy to increase their personal gain. Both of these scenarios, in turn, lead to more centralization within the Bitcoin network, further exacerbating the centralization imposed by mining pools [241]. Meanwhile, the latter issue leads to higher delivery times for honest blocks and opens up the possibility of attacks such as double spending due to forks appearing in the network [242].

The BitTorrent protocol [234], introduced over two decades ago, suffers from a similar issue. As mentioned, in this protocol, the availability of files is also dependent on altruism and reciprocity exhibited by nodes [239]. However, similarly, there is no direct deterrent against nodes that do not contribute to the availability of files in the network. This behavior was referred to as *free-riding*, characterizing nodes that only receive files and do not aid in sharing them with the network [243]. Exhibiting this behavior is not an optimal strategy for nodes, as it decreases the overall availability of files and thus the health of the network.

We argue that both of these examples are specific instances of a more general problem that we label as *selfish behavior*. It refers to any type of behavior exhibited by nodes in peer-to-peer systems where they withhold resources (data) for personal gains even when it is detrimental to the overall health of the system. In the case of Bitcoin, this manifests itself as the withholding of blocks or transactions, while in BitTorrent, it manifests as *leeching*: downloading but not sharing files with others. Figure 5.1 visualizes this concept: honest nodes aid the network by sharing information (e.g., transactions or files), whereas selfish nodes merely reap the benefit of the services provided to them by the network while either sharing nothing or sharing selectively with colluding nodes.

Furthermore, we argue that while this problem is prominent in the Bitcoin and BitTorrent protocols, it is not unique to them. Rather, it is a fundamental problem that all peer-to-peer systems are susceptible to. This problem is rooted in the absence of efficient mechanisms for the accountability of peers at the networking layer: in both instances, a lack of sharing information or data does not directly lead to negative consequences for the offending peer, though they may in the long term as they essentially poison the network by deteriorating its performance (e.g., the availability of files).

Proposed solutions (e.g., see PeerReview [87] and LiFTinG [244]) show promising results with respect to identifying malicious nodes. However, they introduce hefty overheads



(a) Honest (b) Selfish

Figure 5.1: The behavior of nodes.

by relying on the analysis of all outgoing and incoming messages, the requirement of interactions with all nodes within a network, or by being too tailored to be applicable to any type of peer-to-peer system. There is also another largely unaddressed issue prominent in all permissionless networks: the Sybil attack. Enabled by the fact that it is trivial for selfish peers to re-enter the collaborative network under a new identity on multiple instances after abuse of a system [96]. Without active consideration of this attack, any solution becomes vulnerable to manipulation. As such, there is a need for an accountability mechanism that can: 1) exclude nodes that consistently abuse the network, addressing the risk imposed by Sybils; and 2) functions in any peer-to-peer system without introducing unnecessary overhead or complexity.

To address these needs, we propose an accountability system that is based on an evolutionary mechanism referred to as *indirect reciprocity* [245, 246]: a more advanced mechanism rather than the *tit-for-tat* strategy [13]. Our mechanism takes indirect contributions into account when assessing the trustworthiness of nodes. This mechanism also does not rely on global trust scores but accounts for the trustworthiness of peers with their local connections. We argue that *locality* is both necessary and sufficient in achieving sustained cooperation for peer-to-peer networks. In order to mitigate *selfish behavior* and promote fair resource allocation, our approach takes the *subjective contributions* of nodes into account, allowing us to overcome issues of misreporting and manipulation by Sybils. Furthermore, we make use of the MeritRank algorithm [247] to assign subjective trust scores to nodes.

To summarize, our work makes the following contributions:

1. We present a generic accountability mechanism based on indirect reciprocity, which is able to function in any type of peer-to-peer network.

2. We evaluate this mechanism by analyzing its performance through a simulation of the Bitcoin network and a deployment on the network of a BitTorrent client [117], showcasing how our mechanism rewards well-performing peers based on subjective contributions and handles reputations based on indirect reciprocity.

The remainder of this paper is structured as follows: section 5.2 describes the system model in which we operate and formulates the problem we are solving. Next, section 5.3 discusses our design and in section 5.4 we perform a performance analysis of our proposed mechanism. Finally, section 5.5 discusses related work and section 5.6 draws conclusions about our contributions.

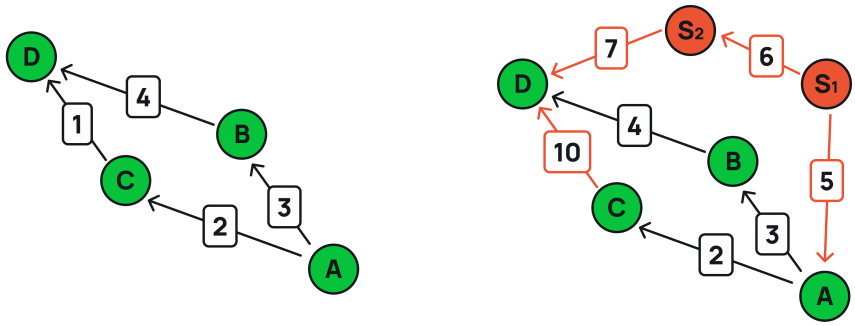


Figure 5.2: An example of a contribution graph G (a) and (b) a modified graph G' under a misreport attack with fake contribution edges and Sybil identities S_1 and S_2 (in red).

5.2 System Model

5

In this work, we consider a peer-to-peer network consisting of N peers with open participation. We assume communication channels are established in a decentralized manner, with each peer available for discovery and connection. Furthermore, the channels between peers are unreliable and unordered, similar to the UDP communication model. Consequently, there is no upper bound on message arrival times, and outbound messages may not successfully reach their intended destinations at all. In addition, each peer possesses a cryptographic keypair, of which the public key uniquely identifies the peer within the network and the private key is used to cryptographically sign outgoing messages. Peers interact with each other by exchanging messages according to a reference protocol, such as the Bitcoin network protocol.

5.2.1 Problem Description

We consider the challenge of constructing a universal protocol to maintain network cooperation despite the presence of peers exhibiting selfish behavior, referred to as *selfish peers*. These peers attempt to optimize their personal gain by reaping the benefits of a system while providing little or no actual value to it, thus undermining the cooperative foundation of a peer-to-peer system.

To mitigate selfish behavior, we consider the contributions made to the network by peers, referred to as *utility*. However, due to the inherent complexity of the problem, relying solely on globally verifiable and accurate proofs of each peer's usefulness in a globally-spanning peer-to-peer network is infeasible. Thus, to preserve the decentralized nature, we rely on aggregating *subjective acknowledgments* of utility generated by peers in the network.

The subjective contributions of peers are modeled as a directed weighted graph referred to as a *contribution graph* $G = (V, E, w)$, consisting of a set of nodes V and a set of edges E. Each edge $e_{ij} \in E$ is associated with a weight w_{ij} defined by the function $w : V \times V \to \mathbb{R}_{\geq 0}$. A weight w_{ij} corresponds to the cumulative contributions made by v_j that are helpful to v_i, where $\{v_i, v_j\} \subseteq V$. An example of a contribution graph can be seen in Figure 5.2a, depicting the scoring assigned to nodes based on the subjective contributions of utility.

Using this definition, we can define a cooperation score $C_k^G \in \mathbb{R}_{\geq 0}$ for each $v_k \in V$. This score quantifies the utility of peer v_k with respect to the entire network, created through an aggregation of all cumulative contributions performed by peer v_k for any other peer, i.e., $w_{lk} \mid l \in V$. This score is used in mechanisms that rely on global trust scores and can be used to punish selfish peers.

Punishing Selfish Peers. In order to maintain the cooperative nature of the peer-to-peer network, it is essential to identify and punish selfish peers. To achieve this, we introduce a punishment mechanism that relies on the cooperation score calculated for each peer. A selfish peer in the graph G is punished if their cooperation score C_k^G falls below a certain threshold value, denoted by τ. This approach introduces, however, its own set of challenges.

Misreporting and the Sybil Attack. A naive way to aggregate all subjective acknowledgments (e.g., the average of all weights w_{ij} corresponding to a peer v_j) is susceptible to misreporting. Misreporting refers to deceptive behavior where participants provide false information about their contributions, seeking an unfair advantage. In conjunction with Sybil attacks [96], strategic peers may attempt to fake their contributions to maximize rewards. Even when employing access restriction techniques, strategic peers can still collude and falsely acknowledge non-existent contributions.

Incorporating the possibility of fake contributions with Sybil peers, we introduce a modified graph $G' = (V', E', w')$, where $V' = V \cup V_s$ denotes a new set of nodes containing a set V_s of Sybil identities, $E' = E \cup E_s$ a set of directed edges with $E' \subseteq V' \times V'$. The modified function $w' : V' \times V' \to \mathbb{R}_{\geq 0}$ additionally contains the fictitious utility performed by Sybils and their colluding nodes. Note that, thus, the modified graph G' contains in addition to all nodes and edges in the original graph G, Sybils nodes V_s and misreported fake contributions edges E_s, created either by nodes together with their Sybils, or with other colluding nodes. A schematic example of this is illustrated in Figure 5.2b, depicting the addition of fictitious contributions in a contribution graph. As such, in order to construct a robust protocol that can withstand the challenges posed by misreporting and Sybil attacks, we introduce a requirement for *misreport resistance*.

Misreport Resistance. The goal of this requirement is to identify and punish selfish peers based on their cooperation scores in the modified graph G', even in the presence of fake contributions and Sybil identities. Let $C_k^{G'}$ be the cooperation score of node v_k in the modified graph G'. A selfish peer is punished in the graph G' if their cooperation score $C_k^{G'}$ falls below a certain value τ', where the difference between τ' and the original threshold τ is bounded by a predefined error margin ϵ, i.e., $|\tau - \tau'| \leq \epsilon$.

By incorporating this misreport resistance requirement, we aim to devise a protocol that can effectively maintain the cooperative nature of the peer-to-peer network, even in the face of adversarial behavior involving misreporting and Sybil attacks. The proposed approach

Figure 5.3: The approach underpinning our mechanism.

seeks to accurately evaluate the contributions of peers while minimizing the impact of fake contributions and ensuring that strategic peers cannot gain an unfair advantage over their honest counterparts.

5.2.2 Selfish Peer Starvation Approach

A key aspect of our approach to handling selfish peers is the reliance on the subjective estimation of cooperation scores. Each peer independently evaluates the cooperation score of other peers based on their direct and indirect contributions. We refer to the subjective cooperation score of peer v_i from the point of view of the evaluating peer v_j as C_{ij}^G.

This subjectivity effectively limits the impact of misreporting since a peer is considered cooperative only if it is directly or indirectly helpful to the evaluating peer. Direct utility is represented by the direct edges originating from the evaluating peer in the contribution graph. Indirect utility, on the other hand, is manifested through the existence of a path between the evaluating peer and the peer being evaluated in the contribution graph. Selfish peers will typically have fewer connections, and the few paths leading to them will have lower weights. Their ability to misreport is also severely limited, as they must perform honest work to receive acknowledgments and create paths in the contribution graph.

Our approach, however, also presents a potential downside: it may overlook the genuine work performed by two honest peers that have no connection to the evaluating peer in the cooperation graph. To mitigate this issue and improve network connectivity, honest peers actively participate in numerous interactions throughout the network, thereby ensuring that the honest peers in the cooperation graph remain connected.

As the network evolves, peers gradually begin to favor more reciprocal partners for future interactions. This shift in behavior leads to a natural consequence for selfish peers, who end up with few or no edges, effectively limiting the services they can access. Sybil attacks also become impractical, as the malicious nodes end up in an isolated subgraph, disconnected from other honest peers.

5.3 Solution Overview

We present a practical protocol designed to sustain cooperation in any peer-to-peer network. The primary objective of this design is to achieve the eventual isolation of selfish peers while ensuring that the network remains connected, allowing honest peers to communicate.

5.3.1 Overview

Each peer in our protocol evaluates the relative contributions of its overlay neighbors. Peers are ranked according to their relative contributions to the network. For instance, peers might be ranked based on the utility they provide for a gossip protocol or how reliably they share files with other peers. Hence, the types of utility taken into account are protocol-dependent.

Our design is composed of four mechanisms, which can be summarized to *Select Neighbors*, *Score Neighbors*, *Gossip Certificates*, and *Rank Graph Nodes*. Figure 5.3 visualizes these processes and their interactions within our system. These four mechanisms work together to adjust the peer-to-peer overlay, accounting for selfish peers. Our mechanism proceeds in rounds. As we operate in a fully decentralized setting, the scheduling and the

length of each round are up to individual peers. This round-based approach ensures liveness by connecting with still-unknown peers and lower-rated peers. The details surrounding the selection of these connections are discussed later on in this section. As the first process, after peer discovery, relies on prior knowledge generated through past interactions, as depicted by the feedback loop visible in Figure 5.3, we start by outlining the *Score Neighbors* mechanism.

Score Neighbors Each peer interacts with its locally connected peers, for instance by receiving or sending messages according to some reference protocol. After these interactions, peers store the information about the provided utility. For example, in a file-sharing scenario, the utility might be the number of files shared or the amount of data transferred, with an associated weight reflecting the importance of that contribution. Peers locally store the cumulative utility provided in the form of a *signed certificate*, showing that some peer v_j was helpful to peer v_i with a utility of w_{ij}. This certificate is signed by v_i and shared with v_j.

Gossip Certificates At the same time, each peer periodically runs a pull-based crawler to collect signed certificates from their neighbors. Thus, neighboring peers exchange information about each other by sharing the latest known certificates. Note that merely storing the most recent certificates is sufficient, as they reflect the latest cumulative opinion on the subjective utility provided by a peer.

Rank Graph Nodes Based on the collected and created certificates, each peer can reconstruct part of the contribution graph. For each node in the reconstructed graph, we assign a cooperation score. This is done with the help of the personalized reputation function MeritRank [247].

Select Neighbors The peers are ranked according to their reputation score. Further interactions within the network are adjusted based on these rankings. Peers prioritize interactions with high-ranking peers while filtering out peers with consistently low reputations. As a result, peers can sustain cooperation in the network, effectively isolating selfish peers as a punishment and promoting collaboration.

5.3.2 Create and Gossip Certificates

As peers interact with each other, they evaluate the provided utility. It is important to note that our approach is agnostic to the specific method by which a peer evaluates another peer. However, we do assume that at each round, peers record their evaluation in the form of a numerical value. As mentioned, the utility w_{ij} provided by peer v_j to peer v_i is recorded as a contribution certificate $\texttt{cert}_{\texttt{ij}}$, which can be defined as:

$$\texttt{cert}_{\texttt{ij}} = (pk_i, pk_j, w_{ij}, r_i, sign_i)$$

A tuple denoted by $\texttt{cert}_{\texttt{ij}}$ represents a record of utility w_{ij} provided by peer v_j for the benefit of peer v_i. It includes parameters such as pk_i and pk_j, which serve as identifiers for peers v_i and v_j, respectively. Additionally, the round number r_i is used as a unique counter to show the local round in which the work was completed. Finally, the $sign_i$ parameter represents the cryptographic signature generated using the private key of peer v_i to ensure the authenticity and integrity of the record.

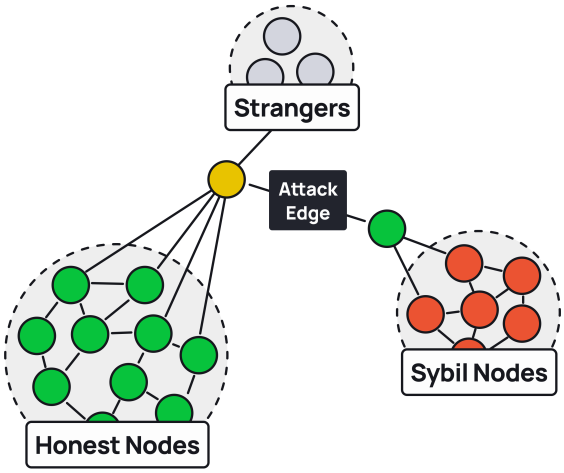


Figure 5.4: An example of a connection graph.

Periodically, peers pull random certificates from their overlay neighbors. Once a certificate for w_{ij} is received, a peer v_k checks its last known certificate for (i, j) and replaces it with the new one, updating the edge weight (i, j) of its locally known contribution graph. Over time, peers will accumulate more certificates, enabling them to construct a more accurate representation of the contribution graph.

5.3.3 Sybil-Tolerant Reputation Ranking

In the local contribution graph, each identifiable node is assigned a ranking that corresponds to its relative contribution to the network. Straightforward approaches that rely on basic global centralized measures, such as total network contribution, are highly susceptible to manipulation. Furthermore, even sophisticated global reputation mechanisms, such as PageRank [236], are in principle not Sybil-tolerant [217]. Consequently, malicious actors can exploit this vulnerability to accrue an undeserved reputation by misreporting the contributions made in conjunction with Sybils.

There exist two types of Sybil attacks: an *active* and a *passive* variant [211]. In the active Sybil attack, any Sybil may have a direct edge to honest peers. Whereas in the passive Sybil attack, there exists a single node, connected to honest peers, with which the Sybils have a direct edge. This latter variant can be seen in Figure 5.4, portraying the connections from the point of view of the node highlighted in yellow. The edge labeled *attack edge* is an instance of a passive Sybil attack. The colluding node in green has numerous connections to the cluster of Sybils with fictitious trust scores.

Personalized reputation mechanisms tackle this problem by attributing a positive reputation to node v_j solely if a path exists from a seed node v_i. This feature intrinsically offers resistance to Sybil attacks, as participants are required to execute tasks to establish a path. We base our reputation algorithm on MeritRank [247], which is a Sybil-tolerant personalized reputation mechanism. The algorithm is based on random walks starting from a source node. The personalized cooperation score of a node v_j from the perspective

of source node v_i on a graph G is the steady-state probability of an α-terminating random walk reaching a node v_j. The parameter α is a transitivity decay, that limits the length of random walks. In practice, we initiate R random walks from an individual node within the cooperation graph. The cooperation score of the C_{ij}^{G} algorithm is determined as the fraction of random walks that start from node v_i and pass through node v_j.

The Sybil tolerance of MeritRank arises from two design choices: the relative personalized ranking and the transitivity decay. The scores are determined based on relative contributions and are periodically recalculated. The score of nodes that stop contributing will naturally decay in comparison to nodes that consistently contribute. This occurs due to the inflation of the global utility generated by nodes within the system. A Sybil attack becomes unfeasible as it necessitates constant genuine contributions in order to obtain positive scores from honest nodes. Transitivity decay, on the other hand, limits the influence of nodes that are distant from the seed node. By incorporating the parameter α, random walks are more likely to terminate as they traverse further from the seed node. Consequently, the reputation score reflects the genuine contribution of a node to the network, as observed from the seed node's standpoint.

This approach prevents Sybil nodes from accumulating high reputation scores by merely connecting to other Sybil nodes or manipulating the structure of the network.

5.3.4 Peer Selection

The peer selection process consists of several stages, with each peer maintaining a dynamic list of selected peers for primary communication during the current round. We employ a specific notation for peer selection, utilizing n slots: a fraction of γ slots are reserved for reputable peers and a fraction of β slots for stranger peers with zero reputation, where $\gamma + \beta = n$. The slots are filled using push and pull strategies, as illustrated in Figure 5.5, and are shuffled after each round r_i.

Initially, a peer proactively fills a predetermined fraction of slots based on the set of known reputable peers and stranger peers, a process referred to as the pull strategy. The peer sends a request to the selected peer, occupying the push slot of the recipient peer. The remaining slots are reserved for the push strategy and are filled by incoming requests. If all push slots are occupied, the incoming request will be rejected. This leads to a system in which both highly-ranked and lower-ranked nodes have the ability to form connections. This counteracts a situation in which only connections to highly ranked nodes are made, as this would lead to an inability for new or low-ranked nodes to gain or improve their reputation.

Bootstrap Period. When a new client joins the network, this node maintains a dynamic list of potential peers, which may be identified through various methods such as querying a centralized directory, distributed hash tables, or gossip protocols. During the bootstrap period, peers thus do not account for reputation to ensure the network remains connected. Specifically, during this period, $\gamma = 0$ and $\beta = n$. Upon the conclusion of the bootstrap period, γ increases and β decreases correspondingly, to account for contributions of nodes documented in the meanwhile.

A typical connection graph is visualized in the aforementioned Figure 5.4. This visualization is performed from the point of view of the node highlighted in yellow. In a typical scenario, a node will favor connections to honest, thus highly reputable nodes.

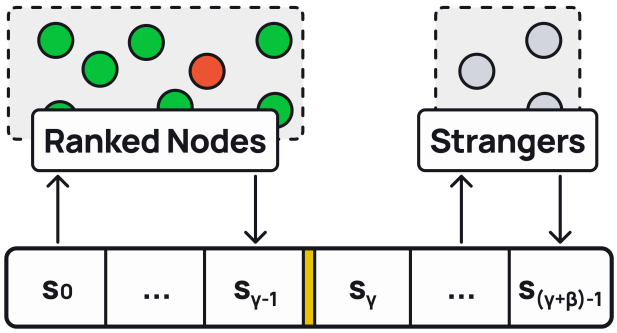


Figure 5.5: The peer selection mechanism with γ slots for reputable peers and β slots for stranger peers.

This is depicted by the high number of connections to that respective cluster. However, strangers must also be given the opportunity to create a reputation for themselves in order to guarantee liveness. As such, our node also has connections with strangers. Finally, this figure also depicts how Sybils can still occur in the system, though they are of minor impact as they require a peer that is reputable to the peer central in this figure, hence, there must still be utility created. Furthermore, their score is bounded by said utility. As such, this visualizes how our system is Sybil-tolerant, as Sybils will never be trusted more than the utility provided by their strongest link through a colluding node. This is visualized by the *attack edge*.

5.4 Experimental results

We provide two experimental setups to evaluate our proposals. First, we simulate a blockchain network based on the latest observed data from the Bitcoin peer-to-peer network. Second, we deploy our accounting mechanism to a client of the file-sharing network of the BitTorrent client Tribler [117]. We report on the data crawled from the network.

5.4.1 Experimental Setup

Bitcoin use case. To demonstrate its practicality, we have applied the mechanism defined in section 5.3 to a simulation of the Bitcoin network. We have simulated numerous aspects of the Bitcoin network in order to benchmark our solution in a realistic setting. The code for this simulation in Python can be found in our repository[1]. We define the utility of a Bitcoin miner in its ability to deliver non-spam transactions and non-stale, non-orphan blocks. Non-spam transactions are transactions that are valid and have a non-zero fee amount. A node v_i creates a certificate for a peer v_j if v_i receives a useful transaction or block from v_j.

We model the experiment after a topology as discussed in [248], which shows that on average 7518 Bitcoin nodes are reachable, and after peer-to-peer latencies as reported in [249]. As such, we use $N = 7518$ for our Bitcoin experiments unless specified otherwise. For MeritRank, we fix the parameter of transitivity decay to $\alpha = 0.2$ and 2000 random walks.

[1] `https://github.com/tribler/bami`.

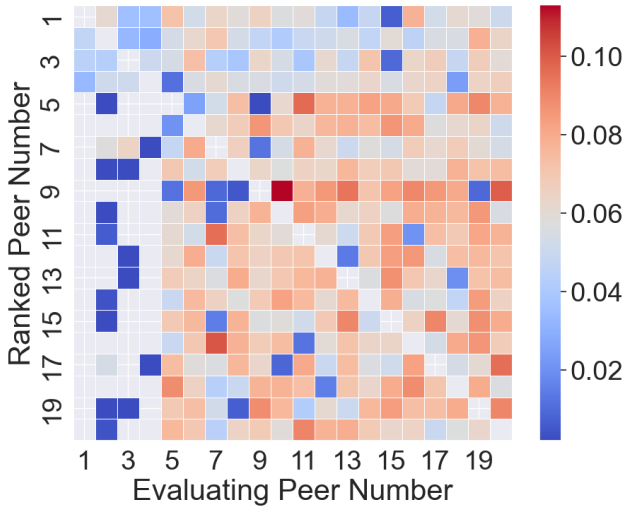


Figure 5.6: A heat map with ranking scores (N=20). Nodes 1-4 are selfish with a share ratio of 0.2.

A Bitcoin transaction is created at some miner node and then relayed over the network to other miner nodes. At the same time, miners run a block creation with an average of 10 minutes by selecting transactions from a local mempool. We simulate a Bitcoin network for 30 days with a transaction rate of 2.5 transactions per second, with all nodes progressing to the next round r_i after each day.

Tribler use case. As free-riding is a well-researched issue in BitTorrent, we have applied an implementation of our accountability mechanism to the Tribler [117] network. Tribler encompasses all logic of the BitTorrent protocol in the form of a file-sharing network. The utility of a peer in Tribler is defined as the amount of bandwidth delivered for the file-sharing and bandwidth-sharing service.

5.4.2 Selfish Peers

We simulate the Bitcoin network with injected selfish peers. We test different *share ratios* for the selfish peers: a factor between 0 and 1 representing the amount of data they share. An honest peer has a share ratio of 1.

Figure 5.6 reports a heat map matrix of the subjective ranking of each peer. We use a small network (N=20) for demonstration purposes. The peers with ids 1-4 are selfish sharing only 20% of the data known to them. The selfish nodes are less preferred than any other nodes by the honest peers, with many selfish peers identified simply as strangers.

We vary the *share ratio* parameter for the selfish peers. We run the simulation with 25% of the network (i.e., 1800 peers) being a selfish peer, each sharing only part of the mempool and blocks. We use the average ranking loss, i.e., the ratio of the average ranking of selfish peers to the average ranking of non-selfish peers. This metric is used to signal the degree to which we can successfully detect and punish selfish peers. We report our findings in Figure 5.7: selfish peers suffer an immediate loss in ranking when they share 10% less than the average honest peer.

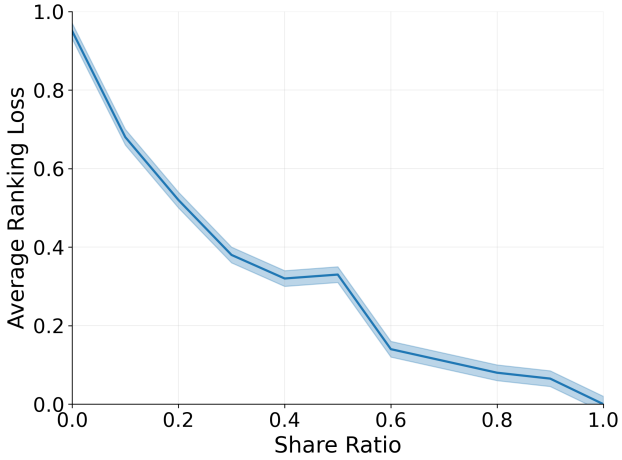


Figure 5.7: Average ranking loss of selfish nodes (N = 7518, with 1800 selfish peers).

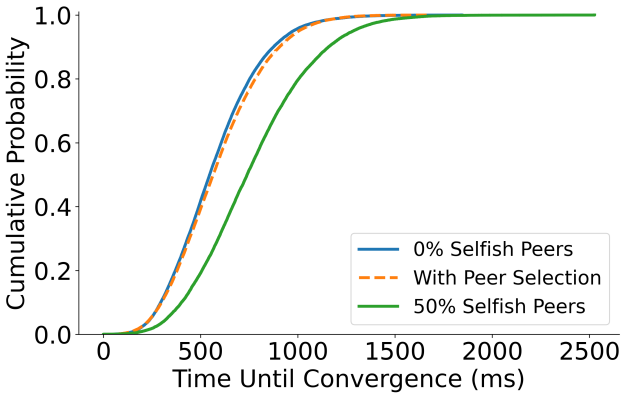


Figure 5.8: The effect of selfish nodes on convergence within Bitcoin.

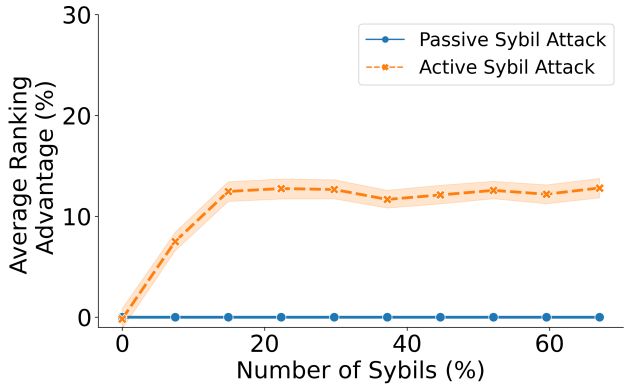


Figure 5.9: The effect of Sybil attacks on the ranking.

Figure 5.8 demonstrates the effect of selfish peers on data convergence. Unsurprisingly, selfish peers can delay the network convergence, delaying each transaction on average by 400 ms. When a peer selection mechanism is employed with $\gamma = 100$ with $n = 125$, we manage to negate the effect of selfish peers with an average delay of 20 ms.

5.4.3 Tolerance to Sybil Attack

To demonstrate the effect of the Sybil attack, we create a Sybil region attempting to manipulate the ranking of one attacker peer. We implement two attack strategies, a passive Sybil attack where each new Sybil peer creates a certificate with one peer and an active Sybil attack where each Sybil creates certificates with each other and the attacker peer. The passive attack can be executed by spamming the peer discovery and introducing fake peers; however, an active attack requires active maintenance of all Sybils to be able to connect to other honest peers.

Figure 5.9 shows the effect of Sybils on the ranking of selfish peers in the network. The x-axis represents the percentage of Sybils in the network, ranging from 0% to 67%. The y-axis shows the average ranking gained, expressed as a percentage. For the active Sybil attack, we report the total advantage gained by all Sybil peers. The passive attack has no effect on the ranking of selfish peers, while the active variant increases the ranking advantage of selfish peers up to a point. Specifically, the selfish peer together with its Sybils is preferred at an average 12% higher than deserved.

The result shows that nodes will not gain any advantage by naively manipulating the ranking through misreporting. To gain any advantage, nodes need to execute a mass-scale attack, effectively maintaining a subnetwork of Bitcoin peers.

5.4.4 Overhead

We introduce only marginal overhead on the bandwidth and memory. One certificate is 220 bytes; as such, after a month's time, a network of $N = 7518$ requires the storage and exchange of up to 10 MBytes of certificates. This is enough to ensure reasonable connectivity and a correct run of the Bitcoin peer.

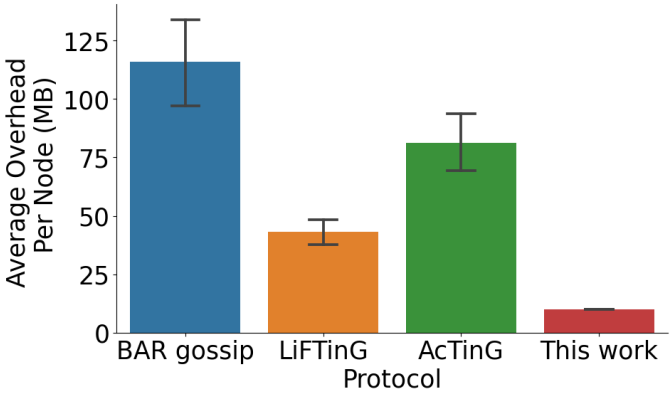


Figure 5.10: The overhead introduced by the protocols.

The number of walks used for MeritRank controls the accuracy of the ranking estimation in exchange for performance. However, the choice of 2000 walks gives an accurate enough estimate and requires only 100 milliseconds for a graph with 7518 nodes.

We conducted a comparative analysis of our protocol against related work, namely BAR gossip[250], LiFTinG[244], and AcTinG[251]. In our simulations, the selfish nodes had a share ratio of 0.2. We set the fanout to 8 with a gossip period of 500 ms. For LiFTinG, we fixed the number of monitors for scorekeeping at $M = 25$. In the case of AcTinG, partners audited each other during the connection with a probability of 15%, in accordance with the recommendation in the original paper [251].

We simulated the protocols using the Bitcoin dataset and measured the average bandwidth overhead per node over 10 runs, as shown in Figure 5.10. Both BAR gossip and AcTinG resulted in significant bandwidth overhead due to the necessity of sharing and auditing the entire history log. LiFTinG, on the other hand, conserved bandwidth by only cross-validating the message. Our protocol introduced the least overhead, as it only required periodic certificate exchanges to detect selfish nodes.

5.4.5 Tribler Deployment Results

Figure 5.11 presents the degree distribution of the contribution graph, which was collected over a year-long period from April 2022 to April 2023. The contribution graph comprises 27,259 nodes and 204,927 edges. The majority of peers interacted with no more than 10 peers, with a median of 1 peer. However, we identified 781 peers with a degree exceeding 100. A small number of peers exhibited an exceptionally high degree of 446 and shared 13 TB of data. These observations align with previous reports of a reliance on a small number of altruistic nodes [239].

We observed that many peers have only a few connections to each other, without forming a connected component. Specifically, the largest connected component comprises 7382 nodes. However, due to the natural skewness of the contribution and the properties of MeritRank, a node can rank only between 600 and 1000 nodes on average. This observation highlights a potential limitation of our approach, suggesting the need for interaction

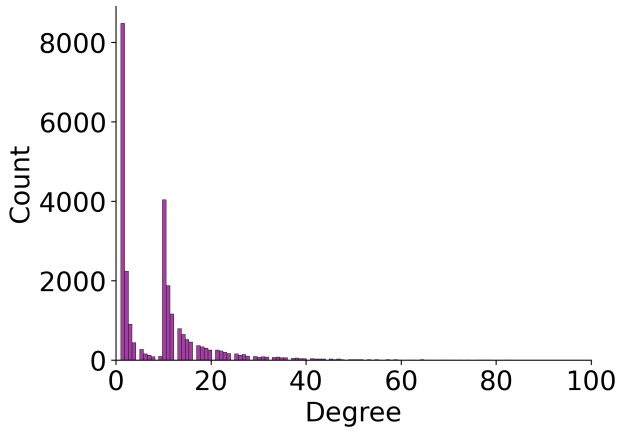


Figure 5.11: Degree distribution of Tribler contribution graph.

with more nodes to enter the ranked connected subgraph. We defer the discussion on incentivizing users to form a bigger connected subgraph to future work.

5.5 Related Work

An adjacent research field to this work is that of accountability in decentralized systems. ConTrib [254] is a mechanism for tracking resource usage across different systems. It prevents abuse by recording resource contributions and consumption in personal ledgers. Parties document these interactions in a cooperative manner: a client proposes a record encapsulating the contribution or consumption of a certain resource in a specific system. The receiver then acknowledges this proposal through a confirmation message. In contrast to our proposed system, ConTrib actively attempts to document abuse and relies on multicasting during both phases of documenting interactions and through pull-based exchanges. As a result, our design reduces the overall message complexity by documenting only merit and does not require agreement.

Table 5.1: Comparison with Related Work

	Sybil Tolerant	Score-Based Reputation	Non-Global view	No Trusted Third Party
This work	✓	✓	✓	✓
LiFTinG [244]	✗	✓	✗	✓
AcTinG [251]	✗	✗	✗	✓
PAG [252]	✗	✗	✗	✓
PeerReview [87]	✗	✗	✗	✓
FullReview [253]	✗	✗	✗	✓
ConTrib [254]	✗	✓	✓	✓
BAR [250]	✗	✗	✓	✗

BAR Gossip [250] is a byzantine, altruistic, and rational tolerant gossip protocol designed for live event streaming. Apart from receiving data directly from a broadcaster, clients receive data through *optimistic push* and *balanced exchange*. In the former mechanism, clients push data in the hope that receiving clients will eventually return the favor. In the latter variant, clients both exchange an equal amount of updates. Clients that divert from the protocol are punished through a proof of misbehavior, leading to a subsequent eviction from the network. BAR relies on the sharing of both the private and the public key with the sole broadcaster.

PeerReview [87] proposes a generic accountability mechanism for distributed systems, in which all messages sent and received by a node are recorded to detect deviations from the protocol. Misbehaving nodes can become suspected and subsequently exposed. A node becomes suspected when it does not acknowledge a certain message. It can exonerate itself from suspicion by acknowledging the initial message. Nodes that are exposed for their misbehavior permanently no longer receive new data. Similarly, FullReview [253] additionally targets selfish nodes through the use of game theory. [255] proposes a methodology for auditing through Accountable Virtual Machines, relying on the analysis of execution logs. These mechanisms, while highly accurate, impose significant overhead when applied to any type of peer-to-peer system.

LiFTinG [244] introduces a protocol designed to detect free-riders in gossip-based systems. This detection is accomplished through two verification mechanisms: direct and a posteriori. In the direct approach, a gossiping node verifies that all transmitted messages are proposed to the correct number of nodes. In the a posteriori approach, clients are asked to submit their log of past interactions, enabling the validation of expected behavior. Nodes that misbehave can be blamed and eventually expelled based on their reputation score, which is managed by assigned managers for each node. AcTinG [251] proposes a similar system, ensuring zero false positives and addressing the issue of colluding selfish nodes. In AcTinG, all clients record their updates in a secure local log. Other clients may then request these logs, revealing any selfish behavior if data was not forwarded according to the protocol. PAG [252] suggests a privacy-preserving accountability mechanism. Each node has a set of monitors that verify whether the specific node correctly gossips data to the expected nodes. This verification is carried out through the validators of the receiving nodes. LiFTinG, AcTinG, and PAG specifically target gossip-based content dissemination protocols, making them not directly applicable to all distributed systems.

Table 5.1 contrasts our work with the related work discussed earlier, based on four criteria we consider essential for decentralized networks. Sybil tolerance is crucial to counteract attacks where peers manipulate their scores through Sybils. We argue that tolerating selfish nodes is more advantageous than entirely eliminating them. A score-based reputation system, as opposed to a binary one, allows for a more nuanced evaluation and enables diverse punishment strategies. Dependence on a global view (e.g., full membership) is impractical for permissionless systems and, depending on the size, may be unfeasible. Therefore, we favor solutions that operate without requiring full network visibility or full network reachability. Lastly, reliance on trusted third parties introduces potential attack vectors, leading to privacy concerns and collusion risks. As shown, our work is the first to satisfy all of these criteria.

5.6 Conclusion

We have presented our protocol for sustaining cooperation in peer-to-peer networks. Our work can function in any type of peer-to-peer system and isolates selfish peers who abuse the network. It does so while introducing minimal overhead and minimizing the risk posed by Sybils. The crucial aspect of our design is *locality*, which enables indirect reciprocity between peers in localized neighborhoods. Peers acknowledge each other's trustworthiness through the mechanism of acknowledgment of performed work. This paradigm allows the design to be Sybil-tolerant. Maximum trust in Sybil nodes is capped by the value of utility they can generate through a colluding bridge node that connects honest clients with Sybils. In any other scenario, Sybils must first generate utility before they are considered trustworthy, thereby negating the very conditions that enable the Sybil attack to be executed.

Our approach works through four mechanisms: selecting reputable peers, scoring peers based on interactions, gossiping said scores through cryptographically verifiable certificates, and ranking nodes based on these certificates and the MeritRank algorithm [247]. Together, they generate a level playing field, in which even unranked peers have the ability to rise to the level of the most reputable peers through our slot-based approach, which is adjusted in a round-based fashion.

Our experiments on the Bitcoin blockchain and the Tribler [117] network demonstrate how our accountability mechanism elegantly forms clusters of connections with reputable peers, which provide utility to the overall network while isolating selfish and malicious nodes. Our results show low time to convergence even with half of the selfish peers. While introducing negligible overhead on bandwidth and memory. Furthermore, our approach is generic enough to allow for applications in any type of peer-to-peer network.

6

Conclusion

This thesis has sought to address the challenges to the current foundation of Web3 by investigating its key shortcomings in reliability, accountability, Sybil attack resilience, and incentive compatibility. We introduced four research questions to guide our study, each corresponding to one of these critical areas. Here, we summarize our conclusions, one for each research question, and provide insights into how our findings contribute to establishing a trustworthy foundation for Web3. We then discuss open questions for further research in this field.

6.1 Conclusions

The conclusions for our supporting research questions are as follows:

1. *Reliability (RQ1):* In Chapter 2, we presented the benchmarking framework Gromit for analyzing the performance and scalability of blockchain systems. Our study provides insights into real-life bottlenecks in terms of throughput and latency, highlighting the importance of realistic benchmarking to understand blockchain performance in real-world scenarios. We conclude that many current implementations of blockchain systems experience performance and availability degradation as the number of peers increases or under real-world conditions.

2. *Accountability (RQ2):* In Chapter 3, we introduced the LØ pre-consensus protocol to address the lack of accountability among blockchain validators. By ensuring miners log all received transactions into a secure mempool data structure, LØ enhances transparency and detects transaction manipulations such as censorship, reordering, or injection. We conclude that accountability mechanisms, such as LØ, are essential for preventing manipulative behaviors and maintaining the integrity and trustworthiness of blockchain networks.

3. *Sybil Attack Resilience (RQ3):* In Chapter 4, we developed MeritRank, a novel Sybil-tolerant feedback aggregation mechanism for reputation systems. Instead of preventing Sybil attacks outright, MeritRank limits the benefits attackers can derive from such attacks. We demonstrated that MeritRank effectively reduces the impact

of Sybil attacks, ensuring that attackers gain benefits only proportional to their genuine contributions. We conclude that reputation systems like MeritRank can enhance the resilience of Web3 ecosystems by limiting the effectiveness of Sybil attacks and ensuring fair assessments of participant behavior.

4. *Incentive Compatibility (RQ4):* In Chapter 5, we designed an incentive-compatible transaction dissemination protocol for permissionless blockchains. Our protocol aligns individual incentives with collective goals, encouraging cooperative behavior while detecting and isolating freeriders and selfish peers. We conclude that Coop is a simple yet effective incentive mechanism based on reputation and accountability that can successfully address selfish peers.

We derive the following high-level conclusions based on the four conclusions:

5. *Holistic Approach to Web3.* A holistic approach to Web3, which considers the interdependencies between various system layers, allows for the identification and mitigation of weak spots in current blockchain systems. This approach has revealed critical issues such as performance bottlenecks (Conclusion 1), lack of transparency in transaction dissemination (Conclusion 2), and vulnerabilities to Sybil attacks and selfish behavior in tokenomics schemes (Conclusions 3 and 4). By adopting a comprehensive perspective, we can develop more robust, scalable, and secure blockchain networks.

6. *Soft-Security Model for Web3.* Hard security models use technical controls like cryptographic algorithms and firewalls, while soft security models leverage social-based controls such as trust networks and reputation mechanisms. Although under-explored, soft security models offer significant potential when combined with hard security models to address challenges in Web3 systems. Incorporating soft security mechanisms, such as accountability protocols before the consensus phase, enhances blockchain integrity and trustworthiness (Conclusion 2). Implementing Sybil-tolerant reputation systems mitigates attacks and ensures fair participant assessments (Conclusion 3). Additionally, incentive-compatible protocols detect and isolate selfish peers, promoting cooperative behavior (Conclusion 4).

6.2 Future directions

This thesis has provided a detailed analysis of the challenges to the foundation of Web3. However, Web3 as technology is still being developed and there is much more to be explored beyond the scope of this work. We now provide several promising directions for future research based on the chapters of this thesis:

1. In Chapter 2, we introduced the generic blockchain benchmarking framework Gromit and conducted a study to reveal the performance limitations of some state-of-the-art blockchain systems. A natural extension would be to consider other reliability metrics beyond throughput and latency, such as the stability of fees, energy efficiency, and resilience under different network conditions. Furthermore, integrating Gromit with predictive analytics could provide deeper insights and proactive optimization

recommendations. The research question we pose is: *"How can benchmarking frameworks be extended to include comprehensive reliability metrics and provide predictive insights for optimizing blockchain systems under diverse real-world conditions?"*

2. In Chapter 3, we introduced the LØ protocol to enhance accountability at the pre-consensus layer. An interesting direction for future research would be to use similar architecture and mechanisms to build a scalable blockchain system while achieving accountability. This could involve exploring the integration of LØ with advanced cryptographic techniques and decentralized identity solutions to further enhance security and transparency. The research question we pose is: *"How can scalable blockchain systems integrate pre-consensus accountability mechanisms with consensus protocols to enhance security and transparency?"*

3. In Chapter 4, we developed MeritRank, a Sybil-tolerant feedback aggregation mechanism. While MeritRank effectively limits the benefits attackers can gain from Sybil attacks, a downside is the potential loss of informativeness due to information decay. Future research could explore ways to improve the informativeness without compromising Sybil-tolerance, such as incorporating machine learning techniques to better assess and maintain the value of information over time. The research question we pose is: *"How can reputation systems balance the trade-off between informativeness and Sybil-tolerance, possibly through advanced machine learning techniques?"*

4. In Chapter 5, we introduced an incentive-compatible transaction dissemination protocol called Coop. Future research could investigate the applicability of such mechanisms beyond peer-to-peer markets, exploring their potential in broader decentralized ecosystems. Additionally, studying whether these mechanisms can achieve desirable properties, such as the preservation of public goods or natural resistance to monopolies, would be valuable. The research question we pose is: *"How can incentive-compatible mechanisms be applied to broader decentralized ecosystems, and what properties can they achieve in terms of public goods preservation and monopoly resistance?"*

Bibliography

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`. Accessed: 2024-03-13.

[2] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. `https://ethereum.org/en/whitepaper/`. Accessed: 2024-03-13.

[3] Alex Murray, Dennie Kim, and Jordan Combs. The promise of a decentralized internet: What is web3 and how can firms prepare? *Business Horizons*, 66(2):191–202, 2023.

[4] Paul E. Ceruzzi. *A History of Modern Computing*. The MIT Press, 2003.

[5] Frederick P Brooks Jr. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[6] Paul Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, 1964.

[7] J.C.R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11, 1960.

[8] Eric A Brewer. Towards robust distributed systems. In *PODC*, 2000.

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*. USENIX Association, 1999.

[11] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66. Springer Berlin Heidelberg, 2001.

[13] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[14] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160. ACM, 2001.

[15] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS*, pages 205–216. Springer, 2005.

[16] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference*, volume 740, pages 139–147. Springer, 1992.

[17] Timothy J. Berners-Lee. Information management: A proposal. Technical report, CERN, 1989. `http://info.cern.ch/Proposal.html`. Accessed: 2024-03-13.

[18] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1999.

[19] Tim Berners-Lee. Html tags, 1992. `https://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html`. Accessed: 2024-03-13.

[20] Jesse James Garrett. Ajax: A new approach to web applications, February 2005. `https://adaptivepath.org/ideas/ajax-new-approach-web-applications/`. Accessed: 2024-03-13.

[21] Shoshana Zuboff. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. PublicAffairs, 2019.

[22] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[23] Samer Hassan and Primavera De Filippi. Decentralized autonomous organization. *Internet Policy Review: Journal on Internet Regulation*, 10(2):1–10, 2021.

[24] International Organization for Standardization. Information processing systems - open systems interconnection - basic reference model. Standard ISO/IEC 7498-1:1984, ISO, 1984.

[25] Optimism PBC. Optimistic rollup: Scalable and instant blockchain transactions, 2020. `https://optimism.io`. Accessed: 2024-03-13.

[26] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370, 2018.

[27] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Whitepaper, 2016. `https://lightning.network/lightning-network-paper.pdf`. Accessed: 2024-03-13.

[28] Etherscan. Etherscan: The ethereum blockchain explorer. https://etherscan.io. Accessed: 2024-03-13.

[29] Blockchain.com. Blockchain.com explorer. https://www.blockchain.com/explorer. Accessed: 2024-03-13.

[30] Till Neudecker and Hannes Hartenstein. Short paper: An empirical analysis of blockchain forks in bitcoin. In *Financial Cryptography and Data Security*, pages 84–92. Springer, 2019.

[31] Robert Leshner and Geoffrey Hayes. Compound: The money market protocol, 2019. https://compound.finance/documents/Compound.Whitepaper.pdf. Accessed: 2024-03-13.

[32] Aave Team. Aave protocol whitepaper, 2020. https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf. Accessed: 2024-03-13.

[33] Protocol Labs. Filecoin: A decentralized storage network, 2017. https://filecoin.io/filecoin.pdf. Accessed: 2024-03-13.

[34] Golem Team. The golem project: Crowdfunding whitepaper, 2016. https://golem.network/doc/Golemwhitepaper.pdf. Accessed: 2024-03-13.

[35] Nicolas Stifter, Aljosha Judmayer, Philipp Schindler, Alexei Zamyatin, and Edgar Weippl. Agreement with satoshi–on the formalization of nakamoto consensus. Cryptology ePrint Archive, 2018. https://eprint.iacr.org/2018/643. Accessed: 2023-09-30.

[36] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains: (a position paper). In *Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[37] Ethereum Foundation. Ethereum's roadmap, 2020. https://ethereum.org/en/roadmap/vision/. Accessed: 2024-03-13.

[38] Eric Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

[39] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[40] Kevin Alarcón Negy, Peter R Rizun, and Emin Gün Sirer. Selfish mining re-examined. In *Financial Cryptography and Data Security*, pages 61–78. Springer, 2020.

[41] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's Peer-to-Peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[42] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *SIGSAC*, pages 3–16, 2016.

[43] Tim Swanson. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems, 2015. https://the-blockchain.com/wp-content/uploads/2016/04/Permissioned-distributed-ledgers.pdf. Accessed: 2024-03-13.

[44] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec*, pages 112–125. Springer, 2016.

[45] David Schwartz, Noah Youngs, and Arthur Britto. The ripple protocol consensus algorithm, 2014. https://ripple.com/files/ripple_consensus_whitepaper.pdf. Accessed: 2024-03-13.

[46] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys*, 2018.

[47] EOS. Eos.io technical white paper, 2018. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. Accessed: 2024-03-13.

[48] Wellington Fernandes Silvano and Roderval Marcelino. Iota Tangle: A cryptocurrency to communicate internet-of-things data. *Future Generation Computer Systems*, 112:307–319, 2020.

[49] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 107:770–780, 2020.

[50] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61, 2019.

[51] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts, 2017. https://plasma.io/plasma-deprecated.pdf. Accessed: 2024-03-13.

[52] Coinbase. https://www.coinbase.com/. Accessed: 2024-02-06.

[53] Stellar. https://www.stellar.org/. Accessed: 2024-02-06.

[54] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, 2012. https://www.peercoin.net/read/papers/peercoin-paper.pdf. Accessed: 2024-03-13.

[55] Dominic Grandjean, Lioba Heimbach, and Roger Wattenhofer. Ethereum proof-of-stake consensus layer: Participation and decentralization. *arXiv preprint arXiv:2306.10777*, 2023.

[56] Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 406–420, 2023.

[57] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloxroute: A scalable trustless blockchain distribution network whitepaper. *IEEE Internet of Things Journal*, 2018.

[58] Ethereum Foundation. Rlpx protocol. `https://github.com/ethereum/devp2p/blob/master/rlpx.md`, 2023. Accessed: 2023-12-13.

[59] Waku. `https://waku.org`. Accessed: 2023-12-13.

[60] Infura: Ethereum api. `https://infura.io`. Accessed: 2023-12-13.

[61] Alchemy. `https://www.alchemy.com`. Accessed: 2023-12-13.

[62] Quicknode. `https://www.quicknode.com`. Accessed: 2023-12-13.

[63] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv:1407.3561*, 2014.

[64] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers, 2016. `https://v1.cosmos.network/resources/whitepaper`. Accessed: 2024-03-13.

[65] Kevin Sekniqi, Daniel Laine, Stephen Buttolph, and Emin Gün Sirer. Avalanche platform, 2020. `https://www.avalabs.org/whitepapers`. Accessed: 2023-09-30.

[66] Hayden Adams. Uniswap v2 core. `https://uniswap.org/whitepaper.pdf`, 2020. Accessed: 2024-03-12.

[67] MakerDAO Team. The maker protocol: Makerdao's multi-collateral dai (mcd) system. `https://makerdao.com/en/whitepaper/`, 2017. Accessed: 2024-03-13.

[68] Alexandra Sims. Blockchain and decentralised autonomous organisations (daos): The evolution of companies? *SSRN Electronic Journal*, 2019.

[69] Aragon DAO. Set your dao governance. `https://aragon.org/how-to/set-your-dao-governance`, 2024. Accessed: 2024-02-06.

[70] Colony. Colony: A platform for community collaboration. `https://colony.io/`. Accessed: 2024-02-06.

[71] OpenSea. Opensea platform. `https://opensea.io`. Accessed: 2023-12-13.

[72] Gang Wang and Mark Nixon. Sok: Tokenization on blockchain. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, pages 1–9, 2021.

[73] Sky Mavis. Axie infinity platform. `https://axieinfinity.com`, 2021. Accessed: 2023-12-13.

[74] Decentraland Team. Decentraland platform. `https://decentraland.org`, 2020. Accessed: 2023-12-13.

[75] Juliane Proelss, Stephane Sevigny, and Denis Schweizer. Gamefi-the perfect symbiosis of blockchain, tokens, defi, and nfts? *Tokens, DeFi, and NFTs*, 2023.

[76] Metamask documentation. `https://docs.metamask.io`. Accessed: 2023-12-13.

[77] Ledger nano user guide. `https://www.ledger.com/academy/hardwarewallet`. Accessed: 2023-12-13.

[78] uport whitepaper. `https://www.uport.me`, 2017. Accessed: 2023-12-13.

[79] Civic whitepaper. `https://www.civic.com`, 2016. Accessed: 2023-12-13.

[80] Jens Ernstberger, Jan Lauinger, Fatima Elsheimy, Liyi Zhou, Sebastian Steinhorst, Ran Canetti, Andrew Miller, Arthur Gervais, and Dawn Song. Sok: data sovereignty. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 122–143. IEEE, 2023.

[81] Brave Software. Brave browser. `https://brave.com/`, 2024. Accessed: 2024-02-06.

[82] Opera. Opera crypto browser. `https://www.opera.com/crypto`, 2024. Accessed: 2024-02-06.

[83] Ethereum Foundation. Web3.js ethereum javascript api. `https://github.com/ethereum/web3.js`, 2015. Accessed: 2023-12-13.

[84] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems - concepts and designs (3. ed.)*. International computer science series. Addison-Wesley-Longman, 2002.

[85] Cryptokitties. `https://www.cryptokitties.co`. Accessed: 2023-12-13.

[86] Bitcoin cash. `https://www.bitcoincash.org/`, 2024. Accessed: 2024-02-06.

[87] Andreas Haeberlen et al. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6):175–188, 2007.

[88] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[89] Bruno Mazorra, Michael Reynolds, and Vanesa Daza. Price of mev: towards a game theoretical approach to mev. In *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, pages 15–22, 2022.

[90] Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, et al. Blockchain censorship. *arXiv preprint arXiv:2305.18545*, 2023.

[91] Tornado Cash. `https://tornado.ws`. Accessed: 2023-12-20.

[92] Mev watch. `https://www.mevwatch.info`. Accessed: 2023-12-20.

[93] Atlas VPN. Hacker attacks on blockchain, 2021. https://atlasvpn.com/blog/blockchain-hackers-stole-3-78-billion-in-122-attacks-in-2020. Accessed: 2024-03-13.

[94] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. *Journal of Cases on Information Technology (JCIT)*, 21(1):19–32, 2019.

[95] Kris Oosthoek. Flash crash for cash: Cyber threats in decentralized finance. *arXiv preprint*, 2021. `https://arxiv.org/abs/2106.10740`. Accessed: 2023-09-30.

[96] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[97] Brian Levine, Clay Shields, and Nathan Margolin. A survey of solutions to the sybil attack. Technical Report Technical Report 2006-052, University of Massachusetts Amherst, 2006.

[98] Leonid Hurwicz. On informationally decentralized systems. *Decision and Organization, edited by C.B. McGuire and R. Radner*, 1972.

[99] Token Engineering Commons. Token engineering fundamentals lecture series, 2023. Accessed: 20 Jul 2025.

[100] Coinmonks. Voting power and governance risks in daos. *Medium*, September 2023. Accessed: 20 Jul 2025.

[101] Tim Copeland. Steem vs tron: The rebellion against a cryptocurrency empire. *Decrypt*, August 2020. Accessed: 20 Jul 2025.

[102] Mitchell Moos. Tron executes hostile takeover of steem, exchanges collude. *Crypto Briefing*, March 2020. Accessed: 20 Jul 2025.

[103] Hanna Halaburda. The hidden danger of re-centralization in blockchain platforms. *Brookings Institution*, April 2025. Accessed: 20 Jul 2025.

[104] Immunefi. Hack analysis: Beanstalk governance attack, april 2022. *Medium*, April 2022. Accessed: 20 Jul 2025.

[105] Rob Behnke. Explained: The beanstalk hack (april 2022). *Halborn*, April 2022. Accessed: 20 Jul 2025.

[106] Anonymous Author. Dao voting mechanism resistant to whale and collusion problems. *Frontiers in Blockchain*, 2024. Accessed: 20 Jul 2025.

[107] United States Department of Justice. Former coinbase insider sentenced in first ever cryptocurrency insider trading case, May 2023. Accessed: 20 Jul 2025.

[108] U.S. Securities and Exchange Commission. Former coinbase manager and his brother agree to settle insider trading charges relating to crypto asset securities, May 2023. Accessed: 20 Jul 2025.

[109] Garrett Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, 1968.

[110] Bulat Nasrulin, Martijn De Vos, and Johan Pouwelse. Code supporting the publication: Gromit: Benchmarking the performance and scalability of blockchain systems, 2025. https://data.4tu.nl/datasets/1d14323c-30ef-489a-bbe0-5d7120313525/1.

[111] Bulat Nasrulin, Georgy Ishmaev, and Johan Pouwelse. Code and dataset supporting the publications. https://github.com/Tribler/bami/tree/master.

[112] Bulat Nasrulin, Georgy Ishmaev, and Johan Pouwelse. Code and dataset supporting the publication: Meritrank: Sybil tolerant reputation for merit-based tokenomics, 2025. https://data.4tu.nl/datasets/b1e7bf95-67d7-4f4e-8043-208e7f71ad84/1.

[113] Ipv8 networking library. `https://github.com/tribler/py-ipv8`. Accessed: 2024-02-06.

[114] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.

[115] Gumby experiment framework. `https://github.com/tribler/gumby`. Accessed: 2024-02-06.

[116] WonderNetwork. Global ping statistics. `https://wondernetwork.com/pings`. Accessed: 2022-05-12.

[117] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 20(2):127–138, 2008.

[118] Caixiang Fan et al. Performance evaluation of blockchain systems: A systematic survey. *Access*, 2020.

[119] Team Rocket et al. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

[120] Marta Lokhava et al. Fast and secure global payments with stellar. In *SOSP*. ACM, 2019.

[121] Bin Cao et al. Performance analysis and comparison of pow, pos and dag based blockchains. *Digital Communications and Networks*, 2020.

[122] Tien Tuan Anh Dinh et al. Blockbench: A framework for analyzing private blockchains. In *ICMD*, 2017.

[123] Hyperledger Caliper. `https://github.com/hyperledger/caliper`, 2018.

[124] Zhongli Dong et al. Dagbench: A performance evaluation framework for dag distributed ledgers. In *CLOUD*. IEEE, 2019.

[125] Harish Sukhwani et al. Performance modeling of hyperledger fabric (permissioned blockchain network). In *NCA*. IEEE, 2018.

[126] Murat Kuzlu et al. Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability. In *IEEE international conference on blockchain (Blockchain)*, 2019.

[127] Marios Touloupou et al. Towards a framework for understanding the performance of blockchains. In *BRAINS*, pages 47–48. IEEE, 2021.

[128] Mohammad Dabbagh et al. Performance analysis of blockchain platforms: Empirical evaluation of hyperledger fabric and ethereum. In *IEEE IICAIET*, pages 1–6, 2020.

[129] Dimitri Saingre et al. Bctmark: a framework for benchmarking blockchain technologies. In *AICCSA*. IEEE, 2020.

[130] Shehar Bano et al. Sok: Consensus in the age of blockchains. In *AFT*, 2019.

[131] Tobias Bamert et al. Have a snack, pay with bitcoins. In *P2P*. IEEE, 2013.

[132] Yossi Gilad et al. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*. ACM, 2017.

[133] Daniel Larimer, Charles Hoskinson, and Stan Larimer. Bitshares: A peer-to-peer polymorphic digital asset exchange, 2018. `https://blog.bitmex.com/wp-content/uploads/2018/06/173481633-BitShares-White-Paper.pdf`. Accessed: 2024-03-13.

[134] Libra white paper. `https://libra.org/en-US/white-paper/`, 2019.

[135] Maofan Yin et al. Hotstuff: Bft consensus with linearity and responsiveness. In *SPDC*, 2019.

[136] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, 2014.

[137] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[138] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[139] Lewis Gudgeon et al. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020.

[140] Apache Kafka. `https://kafka.apache.org`, 2017.

[141] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 2001.

[142] Suporn Pongnumkul et al. Performance analysis of private blockchain platforms in varying workloads. In *ICCCN*. IEEE, 2017.

[143] Jiashuo Zhang et al. Performance analysis of the libra blockchain: An experimental study. In *HotICN*. IEEE, 2019.

[144] BitShares Industrial Performance and Scalability. `https://bitshareshub.io/industrial-performance-and-scalability/`, 2017.

[145] Alex Biryukov et al. Argon2: new generation of memory-hard functions for password hashing and other applications. In *EuroS&P*. IEEE, 2016.

[146] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. Why do my blockchain transactions fail? a study of hyperledger fabric. In *SIGMOD'21*, 2021.

[147] Patrick O'Grady. Preliminary analysis of the invalid minting bug. /urlhttps://medium.com/avalancheavax/preliminary-analysis-of-the-invalid-minting-bug-bee940cbd9e9. Accessed: 2024-03-12.

[148] Michael McSweeney. Stellar hit by transaction issues, developers say 'network is still online' despite node outage. *The Block*, 2021. `https://www.theblockcrypto.com/linked/100649/stellar-network-stoppage-developers-investigation`. Accessed: 2022-05-12.

[149] Rui Wang et al. Performance benchmarking and optimization for blockchain systems: A survey. In *ICBC*. Springer, 2019.

[150] Huma Pervez et al. A comparative analysis of dag-based blockchain architectures. In *ICOSST*, pages 27–34. IEEE, 2018.

[151] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023.

[152] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying Blockchain Extractable Value: How dark is the forest? In *SP*, pages 198–214, San Francisco, CA, USA, May 2022. IEEE.

[153] Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023.

[154] Bruno Mazorra, Michael Reynolds, and Vanesa Daza. Price of MEV: Towards a Game Theoretical Approach to MEV. In *ACM DeFi*, pages 15–22, Los Angeles CA USA, November 2022. ACM.

[155] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *CCS*, pages 692–705, Denver Colorado USA, October 2015. ACM.

[156] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. Extracting godl [sic] from the salt mines: Ethereum miners extracting value. *arXiv preprint arXiv:2203.15930*, 2022.

[157] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice. *arXiv:2212.05111*, 2022.

[158] Flashbots Blockspace Auction. `https://docs.flashbots.net/flashbots-auction/overview`. Accessed: 2024-03-12.

[159] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

[160] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. Bandwidth-efficient transaction relay for bitcoin. *arXiv:1905.10518*, 2019.

[161] Taotao Wang et al. Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain. *IEEE TNSE*, 8(3):2131–2146, 2021.

[162] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *ICDCS*, pages 403–413. IEEE, 2021.

[163] Edward Bortnikov et al. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

[164] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2444–2461. IEEE, 2023.

[165] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. What's the difference? efficient set reconciliation without prior context. *ACM SIGCOMM CCR*, 41(4):218–229, 2011.

[166] Robbert Van Renesse, Dan Dumitriu, Valient Gough, and Chris Thomas. Efficient reconciliation and flow control for anti-entropy protocols. In *LADIS*, pages 1–7, 2008.

[167] Arathi Arakala, Jason Jeffers, and Kathy J Horadam. Fuzzy extractors for minutiae-based fingerprint authentication. In *International conference on biometrics*, pages 760–769. Springer, 2007.

[168] Lum Ramabaja. The bloom clock. *arXiv preprint arXiv:1905.13064*, 2019.

[169] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv:1710.09437*, 2017.

[170] Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Esteves-Verissimo. Repucoin: Your reputation is your power. *IEEE Transactions on Computers*, 68(8):1225–1237, 2019.

[171] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *CCS*, pages 1722–1743, 2021.

[172] Giuseppe Antonio Pierro and Henrique Rocha. The influence factors on ethereum transaction fees. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 24–31. IEEE, 2019.

[173] Alex Auvolat, Yérom-David Bromberg, Davide Frey, and François Taïani. Basalt: A rock-solid foundation for epidemic consensus algorithms in very large, very open networks. *arXiv preprint arXiv:2102.04063*, 2021.

[174] Long Gong et al. Space-and computationally-efficient set reconciliation via parity bitmap sketch (pbs). *arXiv preprint arXiv:2007.14569*, 2020.

[175] Cowswap. `https://cowswap.exchange`. Accessed: 2024-03-12.

[176] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint arXiv:2106.07371v2*, 2021.

[177] Vitalik Buterin. State of research: increasing censorship resistance of transactions under proposer/builder separation (pbs), 2021. `https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance`. Accessed: 2024-03-12.

[178] David Yakira et al. Helix: A fair blockchain consensus protocol resistant to ordering manipulation. *IEEE TNSM*, 18(2):1584–1597, 2021.

[179] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-Fairness for Byzantine Consensus. In *CRYPTO*, volume 12172. Springer International Publishing, Cham, 2020.

[180] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 633–649, 2020.

[181] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 25–36, 2020.

[182] Klaus Kursawe. Wendy grows up: More order fairness. In *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*, pages 191–196. Springer, 2021.

[183] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 316–333. Springer, 2022.

[184] Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint arXiv:2203.11520*, 2022.

[185] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv:2208.00940*, 2022.

[186] Pouriya Zarbafian and Vincent Gramoli. Lyra: Fast and scalable resilience to reordering attacks in blockchains. In *IPDPS*, pages 1–10, 2023.

[187] Haoqian Zhang et al. Flash freezing flash boys: Countering blockchain front-running. In *ICDCSW*, pages 90–95. IEEE, 2022.

[188] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in bft networks. Cryptology ePrint Archive, Paper 2022/898, 2022.

[189] Shutternetwork. `https://github.com/shutter-network/shutter`. Accessed: 2023-01-12.

[190] pmcgoohan. zeromev-geth. `https://github.com/zeromev/zeromev-geth/blob/master/README.md`. Accessed: 2023-01-12.

[191] MEV-boost: Merge ready flashbots architecture. `https://ethresear.ch/t/mev-boost-merge-ready-flashbots-architecture/11177`. Accessed: 2022-04-12.

[192] Cong T Nguyen et al. Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. *IEEE Access*, 7, 2019.

[193] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS International Workshops*, pages 297–315. Springer, 2017.

[194] Aragondao. https://documentation.aragon.org/products/aragon-client/how-to-create-a-dao-using-aragon-client/page-1. Accessed: 2023-03-01.

[195] Coordinape. https://github.com/coordinape. Accessed: 2022-04-01.

[196] SourceCred. https://github.com/sourcecred. Accessed: 2024-03-12.

[197] Alex Rea et al. Colony. technical white paper, 2020. https://colony.io/whitepaper.pdf. Accessed: 2023-03-01.

[198] Pierluigi Freni, Enrico Ferro, and Roberto Moncada. Tokenization and blockchain tokens classification: a morphological framework. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020. https://doi.org/10.1109/ISCC50000.2020.9219709. Accessed: [Your Access Date Here].

[199] Priyeshu Garg. MakerDAO whale with 94% voting power reduces dai stability fee by 4%, 2019. http://bit.ly/4cbvqgb. Accessed: 2022-03-01.

[200] Kelly Liam J. How the juno network DAO voted to revoke a whale's tokens, 2022. https://shorturl.at/nBHNV. Accessed: 2022-04-01.

[201] Phil Daian et al. On-chain vote buying and the rise of dark DAOs, 2018. https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/. Accessed: 2022-04-01.

[202] Stefano Martinazzi and Andrea Flori. The evolving topology of the lightning network: Centralization, efficiency, robustness, synchronization, and anonymity. *Plos one*, 15(1), 2020.

[203] Martin A Nowak. Five rules for the evolution of cooperation. *Science*, 314(5805):1560–1563, 2006.

[204] Jochen Dinger and Hannes Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *First International Conference on Availability, Reliability and Security (ARES'06)*. IEEE, 2006.

[205] Primavera De Filippi et al. Reputation. *Internet Policy Review*, 10(2), 2021.

[206] E.Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized society: Finding web3's soul, 2022. https://ssrn.com/abstract=4105763. Accessed: 2022-05-15.

[207] M. Meulpolder et al. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *IEEE International Symposium on Parallel & Distributed Processing*, 2009.

[208] B. K Liu et al. Personalized hitting time for informative trust mechanisms despite sybils. In *Proceedings of the International Conference on Autonomous Agents & Multi-agent Systems*, 2016.

[209] Lorenzo Alvisi et al. Sok: The evolution of sybil defense via social networks. In *IEEE Symposium on Security and Privacy*. IEEE, 2013.

[210] Bimal Viswanath, Ansley Post, Krishna P Gummadi, and Alan Mislove. An analysis of social network-based sybil defenses. *ACM SIGCOMM Computer Communication Review*, 40(4):363–374, 2010.

[211] Alexander Stannat, Can Umut Ileri, Dion Gijswijt, and Johan Pouwelse. Achieving sybil-proofness in distributed work systems. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1263–1271. International Foundation for Autonomous Agents and Multiagent Systems, 2021.

[212] Kevin Walsh and Emin Gün Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *NSDI*, volume 6, 2006.

[213] Rahim Delaviz et al. Sybilres: A sybil-resilient flow-based decentralized reputation mechanism. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE, 2012.

[214] Ferry Hendrikx et al. Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75, 2015.

[215] Emanuele Bellini et al. Blockchain-based distributed trust and reputation management systems: A survey. *IEEE Access*, 8, 2020.

[216] Stan Gurtler and Ian Goldberg. Sok: Privacy-preserving reputation systems. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2021:107–127, 2021.

[217] Alice Cheng and Eric Friedman. Sybilproof reputation mechanisms. In *ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132, 2005.

[218] K. Hoffman et al. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys (CSUR)*, 2009.

[219] Bimal Viswanath et al. Exploring the design space of social network-based sybil defenses. In *International Conference on Communication Systems and Networks (COMSNETS 2012)*. IEEE, 2012.

[220] Eleni Koutrouli and Aphrodite Tsalgatidou. Taxonomy of attacks and defense mechanisms in p2p reputation systems—lessons for reputation system designers. *Computer Science Review*, 6(2), 2012.

[221] S. Seuken et al. Work accounting mechanisms: Theory and practice. In *Working Paper. Department of Informatics*. University of Zurich, 2014.

[222] Youssef El Faqir et al. An overview of decentralized autonomous organizations on the blockchain. In *Proceedings of the 16th International Symposium on Open Collaboration*. ACM, 2020.

[223] Divya Siddarth et al. Who watches the watchmen? a review of subjective approaches for sybil-resistance in proof of personhood protocols. *Frontiers in Blockchain*, 3, 2020.

[224] Shayan Eskandari et al. SoK: oracles from the ground truth to market manipulation. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, 2021.

[225] Yuxi Cai et al. A truth-inducing sybil resistant decentralized blockchain oracle. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2020.

[226] Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[227] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651, 2003.

[228] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 267–278, 2006.

[229] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 3–17. IEEE, 2008.

[230] Bimal Viswanath, Mainack Mondal, Krishna P Gummadi, Alan Mislove, and Ansley Post. Canal: Scaling social network-based sybil tolerance schemes. In *Proceedings of the 7th ACM european conference on Computer Systems*, 2012.

[231] Discourse. `https://www.discourse.org`. Accessed: 2024-04-01.

[232] Evan Miyazono. Sourcecred: An introduction to calculating cred and grain, 2020. https://research.protocol.ai/blog/2020/sourcecred-an-introduction-to-calculating-cred-and-grain. Accessed: 2022-04-01.

[233] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *ACM Conference on Information-Centric Networking*, pages 1–11, 2019.

[234] Bittorrent. `https://bittorrent.com/`. Accessed: 2023-08-12.

[235] B. Bahmani et al. Fast incremental and personalized pagerank. *VLDB*, 2010.

[236] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. Proceedings of the Seventh International World Wide Web Conference.

[237] MakerDAO-forum. https://forum.makerdao.com. Accessed: 2024-04-01.

[238] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*, pages 515–532. Springer, 2017.

[239] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent. In *Proc. of NSDI*, volume 7, page 4, 2007.

[240] Saeideh G. Motlagh, Jelena Mišić, and Vojislav B. Mišić. The impact of selfish mining on bitcoin network performance. *IEEE Transactions on Network Science and Engineering*, 8(1):724–735, 2021.

[241] Alireza Beikverdi and JooSeok Song. Trend of centralization in bitcoin's distributed network. In *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6, 2015.

[242] Hongyue Kang, Xiaolin Chang, Runkai Yang, Jelena Mišić, and Vojislav B. Mišić. Understanding selfish mining in imperfect bitcoin and ethereum networks with extended forks. *IEEE Transactions on Network and Service Management*, 18(3):3079–3091, 2021.

[243] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM Workshop on Practice and Theory of Incentives in Networked Systems*, PINS '04, page 228–236, New York, NY, USA, 2004. Association for Computing Machinery.

[244] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. Lifting: lightweight freerider-tracking in gossip. In *Middleware*, pages 313–333. Springer, 2010.

[245] Martin A Nowak and Karl Sigmund. Evolution of indirect reciprocity. *Nature*, 437(7063):1291–1298, 2005.

[246] Laura Schmid, Krishnendu Chatterjee, Christian Hilbe, and Martin A Nowak. A unified framework of direct and indirect reciprocity. *Nature Human Behaviour*, 5(10):1292–1302, 2021.

[247] Bulat Nasrulin, Georgy Ishmaev, and Johan Pouwelse. Meritrank: Sybil tolerant reputation for merit-based tokenomics. In *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 95–102. IEEE, 2022.

[248] Matthias Grundmann, Max Baumstark, and Hannes Hartenstein. On the peer degree distribution of the bitcoin p2p network. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2022.

[249] Elias Rohrer and Florian Tschorsch. Blockchain layer zero: Characterizing the bitcoin network through measurements, models, and simulations. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 9–16. IEEE, 2021.

[250] Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 191–204, 2006.

[251] Sonia Ben Mokhtar, Jérémie Decouchant, and Vivien Quéma. Acting: Accurate freerider tracking in gossip. In *SRDS*, pages 291–300. IEEE, 2014.

[252] Jérémie Decouchant, Sonia Ben Mokhtar, Albin Petit, and Vivien Quéma. Pag: Private and accountable gossip. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 35–44, 2016.

[253] Amadou Diarra, Sonia Ben Mokhtar, Pierre-Louis Aublin, and Vivien Quéma. Full-review: Practical accountability in presence of selfish nodes. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 271–280, 2014.

[254] Martijn de Vos and Johan Pouwelse. Contrib: Maintaining fairness in decentralized big tech alternatives by accounting work. *Computer Networks*, 192:108081, 2021.

[255] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, page 119–134, USA, 2010. USENIX Association.

Curriculum Vitæ

Bulat Aydarovich Nasrulin

26/09/1993	Date of birth in Tashkent, Uzbekistan

Education

The following is a list of only higher education followed leading up to this thesis, omitting individual courses and summer schools.

2/2019-2/2024	Ph.D. Student, Distributed Systems Group, Delft University of Technology, The Netherlands, *Trustworthy Foundation for Web3*
9/2015-9/2017	M.Sc. Computer Science, Innopolis University, Russia
9/2011-8/2015	B.Sc. Applied Math, Kazan Federal University, Russia

Open Source Software Contributions

The following is a list of major open source projects contributions, omitting smaller projects.

Project	**Language**	**URL**
Tribler	Python	https://www.tribler.org/
Gumby	Python	https://github.com/tribler/gumby
MeritRank	Rust & Python	https://github.com/Intersubjective/meritrank-rust

List of Publications

1. ***Bulat Nasrulin**, Martijn De Vos, Georgy Ishmaev, Johan Pouwelse*: Gromit: Benchmarking the Performance and Scalability of Blockchain Systems, *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2022.

2. ***Bulat Nasrulin**, Georgy Ishmaev, Johan Pouwelse*: MeritRank: Sybil Tolerant Reputation for Merit-based Tokenomics, *Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2022.

3. ***Bulat Nasrulin**, Rowdy Chotkan, Johan Pouwelse*: Sustainable Cooperation in Peer-To-Peer Networks, *IEEE Conference on Local Computer Networks (LCN)*, 2023.

4. ***Bulat Nasrulin**, Georgy Ishmaev, Jérémie Decouchant, and Johan Pouwelse*: LØ: An Accountable Mempool for MEV Resistance. *ACM Middleware*, 2023.

5. ***Bulat Nasrulin**, Johan Pouwelse*: Decentralized collaborative version control, Proceedings of the 2nd International Workshop on Distributed Infrastructure for Common Good, 2021

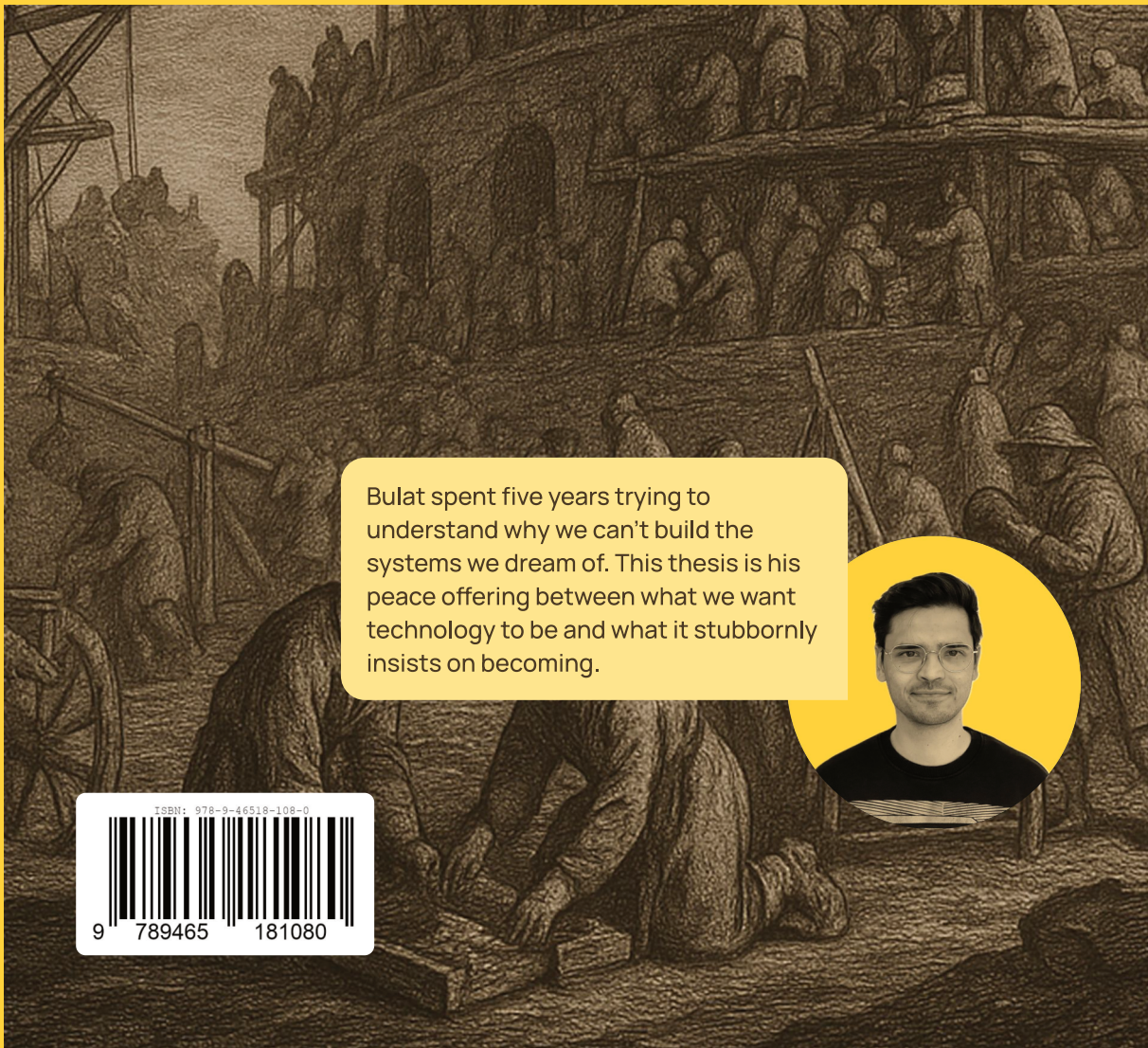
6. *Rowdy Chotkan,* ***Bulat Nasrulin**, Johan Pouwelse*: Interlayer Feedback: Reputations for Spam Prevention in Decentralized Systems, *under review*

Included in this thesis.

Web3 represents an ambitious attempt to democratize access to financial instruments and reimagine economic coordination. At its core, Web3 envisions a public infrastructure where users actively contribute to and share control of the systems they depend upon. Yet the gap between this vision and today's implementations reveals deep architectural flaws.

This thesis revisits Web3's foundations from first principles. Our investigation reveals that persistent vulnerabilities like transaction manipulation, Sybil attacks, and selfish behavior stem from misguided assumptions about participant behavior and misplaced focus on consensus alone.

Trustworthy Web3 foundations must account for the fact that protocols can be forked, validators run different software versions, and participants operate under diverse incentives. These realities demand accountability and incentive alignment mechanisms beyond traditional security models.



Bulat spent five years trying to understand why we can't build the systems we dream of. This thesis is his peace offering between what we want technology to be and what it stubbornly insists on becoming.

ISBN: 978-9-46518-108-0