Decentralized Payment Systems: Principles and Design

Editors: Giulia Fanti and Pramod Viswanath

January 19, 2019

Contributing Authors

Chapters 1 and 2
Giulia Fanti, CMU
Pramod Viswanath, UIUC

Chapter 3
Vivek Bagaria, Stanford
Sreeram Kannan, UW Seattle
David Tse, Stanford
Giulia Fanti, CMU
Pramod Viswanath, UIUC

Chapter 4
Giulia Fanti, CMU
Jiantao Jiao, UC Berkeley
Sewoong Oh, UW Seattle
Pramod Viswanath, UIUC

Chapter 5
Salman Avestimehr, USC
Sreeram Kannan, UW Seattle
Pramod Viswanath, UIUC

Chapter 6
Mohammad Alizadeh, MIT
Giulia Fanti, CMU
Pramod Viswanath, UIUC

Chapter 7
Giulia Fanti, CMU
Leonid Kogan, MIT
Sewoong Oh, UW Seattle
Pramod Viswanath, UIUC

Chapter 8
Giulia Fanti, CMU
Andrew Miller, UIUC
Pramod Viswanath, UIUC

Contents

Chapter 1

Introduction

Giulia Fanti, CMU
Pramod Viswanath, UIUC

The need to build a decentralized trust system is broad and pressing. For centuries, mutually distrustful parties have collaborated to build empires, economies, and social structures. However, these collaborations and interactions have historically been managed with opaque systems that are susceptible to corruption and extreme imbalance of power. For example, consider the social contract between a government and its population. A specific party (the government) is endowed with disproportionate power; if that party is corrupted, it is difficult for the system to revert to a state that is acceptable to all parties. *Decentralized trust systems* are digital systems in which multiple parties can collaborate on specific tasks without requiring parties to trust one another. An example of a task that requires decentralized trust is that of running a payment system. Although decentralized computer systems have traditionally posed substantial technical challenges (e.g., scalability and security in peer-to-peer networks), three main technological trends are facilitating the proliferation of decentralized trust systems today: ubiquitous Internet connectivity, low-cost computing and storage, and the development of blockchain technology. Delivering these new mechanisms of trust—and the wide-ranging possibilities they bring—are of great interest to society at-large.

In this monograph we envision the development of a highly-scalable and fully-decentralized payment system: Unit-e. Decentralized payment systems inherently exhibit a number of desirable properties: (i) the diffusion of control among stakeholders; (ii) the ability to engage in trusted commerce without a centralized intermediary; (iii) the potential to disrupt the rents extracted by centralized intermediaries facilitating commerce; and (iv) global consistency and transparency on a shared ledger. Because of the sensitive nature of money, such a payment system requires decentralization at several layers. The design and development of the system should be decentralized to ensure that no single party controls the implementation of the system. The algorithms governing the system should be decentralized to avoid a single party controlling the flow of money during daily operations. The dissemination of fees should be decentralized to democratize the management of money. And finally, the governance of the payment system should be decentralized to prevent a single party from modifying system parameters and algorithms in their favor. The objective for Unit-e is to satisfy these decentralization constraints in a manner that is both performant and robust.

The original vision for Bitcoin was to build precisely such a payment system. In many ways, it was successful; Bitcoin has achieved high levels of decentralization compared to centralized money supplies, and remarkable levels of awareness for cryptocurrencies, although merchant adoption has been nearly non-existent. Nonetheless, Bitcoin's architectural choices have constrained both

performance and scalability, thereby limiting its utility as an everyday payment unit. This has led to questions about the intrinsic value of both Bitcoin and digital currencies in general [89]. These doubts are primarily based on the present low usage of Bitcoin, which can be addressed by recent proposals to build substantially faster blockchains than Bitcoin's. The scaling improvements have been largely enabled by architectural choices that emphasize scalability—often by choosing centralization, which goes counter to the core tenet of decentralized trust systems. Building a truly decentralized and efficient payment unit is a core challenge, and indeed, despite the plethora of digital tokens in existence today, there is still no candidate that can be used for the regular exchange of value in the daily lives of ordinary people.

Unit-*e* Unit-*e* is a cryptocurrency that specializes exclusively on payments, with a strong emphasis on performance and state-of-the-art, decentralized scalability. Whereas blockchains offering smart contracts must solve the problem of full, decentralized state-machine replication, lightweight payment transactions offer massive parallelism. Consequently, we narrow the objectives of a general purpose blockchain system, and tackle these questions in a holistic, first-principles manner. We note that although focused on payments, we find that much of the research output involved in the design of Unit-*e* also provides programmability (e.g. through smart contracts written in Turing or pseudo-Turing complete languages); we discuss these ramifications in the appropriate sections throughput the manifesto. To become a ubiquitous global payment system, Unit-*e* is designed to meet the following five requirements in a *fully-decentralized* manner:

1. **Security.** The system should prevent unauthorized or invalid payments from being executed.

2. **Latency.** Transactions should be processed seamlessly, on the timescale of seconds.

3. **Throughput.** The network as a whole should be able to confirm up to thousands of transactions per second.

4. **Usability.** The system should be accessible at all times, offer low and predictable fees and a low cost of operating the network and provide a seamless and predictable user experience.

5. **Privacy.** The system should prevent unauthorized parties from accessing transaction logs.

A key challenge is to meet these requirements in an efficient, scalable, decentralized platform.

This manifesto represents the output of such a research engagement and it highlights key insights that are fundamental to the building of a truly scalable decentralized payment system. Specifically, we draw inspiration from information theory, networking, coding theory, game theory, and economics, in addition to more well-known connections to distributed systems and cryptography.

1.1 Requirements

We begin this chapter by revisiting our requirements in greater detail and providing benchmark performance figures. Then, we discuss the technical building blocks needed to deliver the desired figures of merit, formulating basic and fundamental questions of core scientific and engineering relevance to blockchains.

Security. If implemented naively, blockchain systems can introduce many security concerns related to usability, implementation issues, incentives, and fault-tolerance. From an algorithmic design standpoint, we wish to protect against two main types of security violations. The first involves unauthorized users making payments with other users' funds. The second involves users double spending their own money. Both attacks amount to theft, but the second threat is unique to digital money. Assuming a secure codebase, unauthorized transactions are organically prevented through the use of asymmetric cryptography. Double spends, on the other hand, should be specifically prevented by decentralized consensus mechanism. Tackling this challenge in a low-latency, high-throughput system is particularly challenging and a key focus of Unit-e's design.

Low Latency. Latency is critical in a consumer-facing payment system, particularly for point-of-sale transactions. Unit-e therefore aims to achieve confirmation latencies on the order of 15 seconds for on-chain transactions, and 2-4 seconds for off-chain transactions. Although some cryptocurrencies achieve comparable latencies today, they do so at the expense of decentralization. We find that this tradeoff is not fundamental; it is possible to achieve fast confirmation using decentralized algorithms. We sidestep this challenge through a ground-up redesign of consensus mechanisms, coupled with new routing algorithms for payment channel networks.

On the consensus front, we develop new algorithms with strong theoretical performance guarantees, which are able to approach the limits of what is physically possible in a blockchain. Our process for designing such algorithms involves carefully and systematically deconstructing blockchains into their core functionalities and rebuilding them from scratch. A key intuition is that in order to get good throughput and latency, the processes of proposing and confirming blocks should be algorithmically separated. Our design of secure and efficient consensus algorithms is discussed in Chapter 3.

In parallel with fast on-chain proof-of-stake (PoS) consensus, we implement a payment channel network targeted towards high-volume, low-value transactions (e.g. point-of-sale transactions). Payment channel networks are overlay networks that use on-chain consensus to set up escrow accounts (or channels) between pairs of users. By exploiting clever cryptographic constructs, users can route transactions over a path of these channels, even if the two endpoints do not share a channel themselves. The key benefit is that users can verify transactions instantaneously without waiting for confirmation from the blockchain. This significantly reduces confirmation latency compared to on-chain transactions; the main delay stems from passing the transaction to the recipient, which is a fast, point-to-point operation that can take as little as a second for direct channels.

Throughput. A closely-related concept to latency is throughput—the number of transactions processed per second. We are targeting throughputs of 5,000-10,000 transactions per second. For comparison, note that Visa's networks process almost 1,700 transactions per second on average, and an order of magnitude more at its peak [179]. Also for comparison, Bitcoin's current average throughput is estimated between 3.3 and 7 transactions per second, and Ethereum reaches between 10-30 transactions per second. Bridging this large gap is technically nontrivial and requires significant innovation. As an aside, we note that the target throughput metrics are already at the physical limits of a typical modern P2P network; a 20 Mbps network physically cannot handle substantially more transactions per second without making severe compromises (typically in security).

To achieve our target throughput figures, we rely on a combination of novel consensus mechanisms, entirely new ways of sharding, and payment channel networks. We design novel consensus algorithms (Chapter 3) and new peer-to-peer (P2P) networking algorithms that breathe in sync with the informational imperatives of the consensus algorithms (Chapter 4). This allows us to achieve optimal throughput (and latency and security, simultaneously), constrained only by the physical limits of the

	Security	Latency	Throughput	Usability	Privacy
3: Consensus Algorithms	×	×	×	×	
4: Network-aware Consensus	×	×	×	×	
5: Storage and Computation	×		×		
6: Payment Channel Networks		×	×		
7: Economics	×			×	
8: Privacy					×

Table 1.1: Relation between desired properties and the topics in each chapter.

networking layer, for on-chain transactions. We propose new sharding algorithms based on coding the blockchain data which achieve optimal tradeoffs in security, storage, and computation; this is conducted in Chapter 5. Finally, we design new payment channel networks in Chapter 6, where we propose a new routing algorithm that achieves up to 50% higher transaction throughput than state-of-the-art routing proposals, without sacrificing latency.

Usability. Usability is critical to a payment system, since customers must be able to make payments whenever they wish. In principle, the distributed, peer-to-peer network of Unit-e nodes gives some protection against random network fluctuations. In practice, Bitcoin has been extraordinarily reliable over its existence for nearly a decade; measurements have shown a network reliability exceeding 99.9999% since 2011. The incentive mechanisms of Unit-e —discussed in Chapter 7— are designed to motivate and reward participant nodes towards similar levels of availability.

Privacy. Privacy is one of the primary challenges associated with using a cryptocurrency as a real payment system. By design, cryptocurrencies are designed to be transparent; the blockchain is inherently a public, verifiable record of transactions. However, this can lead to significant privacy violations if people are to use the cryptocurrency for everyday transactions. To navigate this tradeoff, Unit-e will incorporate privacy protections against blockchain-level attacks (e.g., Zcash, a privacy-preserving cryptocurrency, provides a starting point of our design). Unit-e also protects against network-level attacks by proposing a novel privacy solution called Dandelion that protects against network adversaries linking users' transactions to their IP addresses (even with the data being encrypted) [34].

1.2 Outline

Our goal is to take a first-principles, full-stack approach to designing Unit-e's payment system. In this manifesto, we provide a comprehensive set of solutions to the key requirements set up formally earlier, as well as setting the stage for continued research on blockchains. Given the vast subject matter we have found it natural to divide the solution space into several chapters. Table 1.1 summarizes the connection between the figures of merit and the different chapters. The chapters cover state of the art research in blockchains conducted under the collective experience and knowledge gained by the diverse disciplines of information theory, coding theory, communication theory, distributed algorithms, economics, and data networking. A summary of the main findings of all the chapters is provided next.

Chapter 2

Unit-e: Summary of Design

Giulia Fanti, CMU
Pramod Viswanath, UIUC

The goal of this section is to summarize the design and research contributions of Unit-e. Whereas the previous section outlined our technical requirements, we begin this section with an abstraction of the architectural components of a cryptocurrency. Each chapter of this manifesto will address one piece of this abstraction through a first-principles, ground-up design.

2.1 The Architecture of Cryptocurrencies

Blockchains are often represented conceptually by a *layered* model, much like the OSI networking model. In designing Unit-e, we use this layer-based model as a starting point, and expand it to include components that are specifically relevant to cryptocurrencies. As shown in Figure 2.1, **Layer 1** technologies refer to the core blockchain; this includes everything from consensus mechanisms to data structures to the networking stack. Traditionally, the bulk of blockchain development and research has addressed layer 1. **Layer 2** instead describes technologies that use an underlying blockchain to build applications. Although layer 2 technologies cannot exist without layer 1, certain layer 2 technologies could prove essential for the long-term viability and scalability of blockchains. Payment channel networks—such as Bitcoin's Lightning network and Ethereum's Raiden network—are prominent exemplar technologies.

The layer abstraction is useful for reasoning about the development and structure of blockchains. However, some aspects of cryptocurrencies span multiple layers and are not easily categorized. Three such aspects are privacy, economics, and governance. *Privacy* refers to the ability of users to make transactions without revealing information about this transaction to other users. Financial systems in particular have stringent privacy requirements, which are not necessarily satisfied by naive blockchain designs. Moreover, privacy requirements exist at both layers 1 and 2, so privacy technologies are not naturally captured by the layer model. Hence we view privacy as encompassing both layers in Figure 2.1. *Economics* refers to the mechanisms that incentivize users to offer storage, computation, and bandwidth needed for daily operations. It also refers to the tenuous relation between cryptocurrencies and fiat currencies, and how to price one in terms of the other. Since economics depend closely on algorithmic decisions regarding layers 1 and 2, as well as the privacy layer, we think of economics as encompassing all of them. Finally, *governance* refers to the processes and rules for making decisions about the network, ranging from technical recommendations to disaster recovery. Governance can affect all aspects of the blockchain, and therefore encompasses

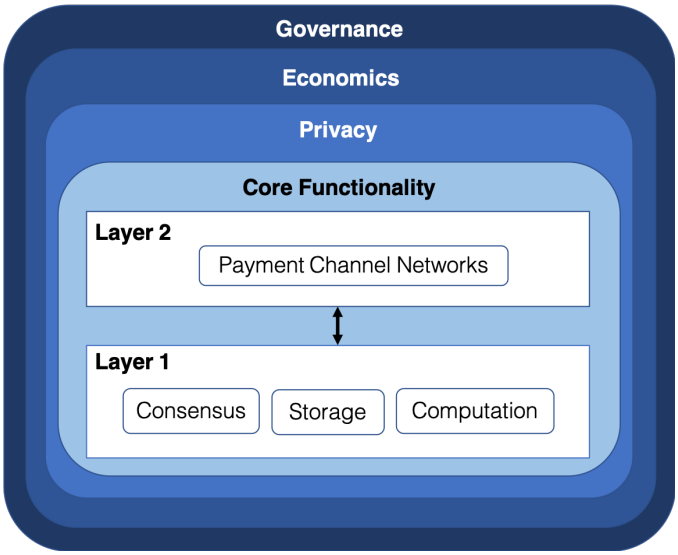


Figure 2.1: Abstract model of cryptocurrencies.

all of the previous components. Although governance is critical to the success of any project, it is not a research topic in the traditional sense, and hence we do not discuss it further in this research manifesto.

2.2 Unit-e's design

Armed with this abstraction, we systematically tackle each component of the architecture. Our manifesto begins with the core layer 1 technologies: consensus, storage, and computation. Next, we cover payment channel networks at layer 2, followed by economics and privacy. For each of these technologies, we first consider what performance levels are possible according to the laws of physics. Next, we propose algorithms that are tailored to meet those physical limits. In the following, we summarize the contents and research contributions of each of these chapters.

Chapter 3: Consensus

For a given security level, two key performance metrics characterize a blockchain consensus protocol: throughput and latency. In a decentralized setting, these measures are limited by underlying physical network attributes—namely, communication capacity and speed-of-light propagation delay. We begin this chapter by showing that existing cryptocurrencies operate far from these physical limits. A natural question is whether *any* consensus algorithm can achieve performance metrics close to the physical limits. We answer this question in the affirmative by presenting Prism, a new blockchain protocol that achieves 1) security against up to 50% adversarial nodes; 2) optimal throughput up to the capacity C of the network; 3) order-optimal confirmation latency for honest transactions proportional to the propagation delay D, with confirmation error probability exponentially small in the bandwidth-delay product CD; 4) eventual total ordering of all transactions. Our approach to the design of this protocol is based on *deconstructing* the blockchain into its basic functionalities and systematically scaling up these functionalities to approach their physical limits. We begin by presenting Prism in the proof-of-work setting, then discuss how to extend the intuitions underlying it to a proof-of-stake consensus algorithm.

Prism builds on the following intuition: to achieve high throughput, a system should produce blocks frequently. However, naively increasing the block production rate leads to forking, which hinders transaction confirmation and reduces the security of the protocol. To resolve this tension, we architecturally separate the act of producing new blocks from the act of confirming them. Other systems have proposed decoupling various aspects of consensus. However, Prism's decoupling in particular leads to a heavily-structured directed acyclic block-graph (DAG) that simultaneously facilitates analysis, while giving optimal latency and throughput guarantees. This DAG is illustrated in Figure 2.2.

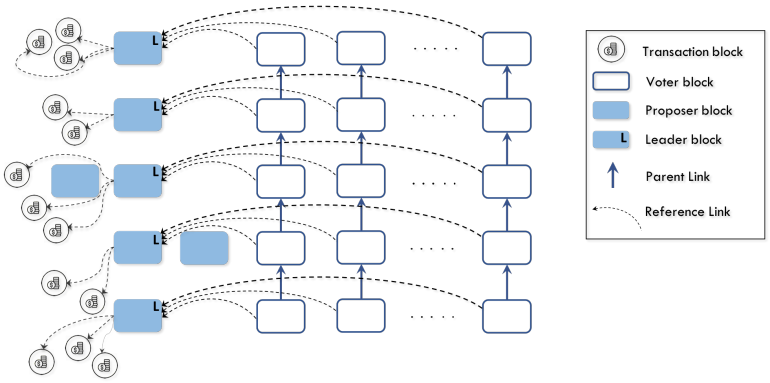


Figure 2.2: Prism is a consensus mechanism that explicitly deconstructs the various roles that blocks play in a blockchain: logging transactions, proposing transactions, and voting on other blocks. Prism achieves optimal transaction throughput and latency, while satisfying strong security guarantees.

Chapter 4: Network-Aware Consensus

In Chapter 3, we proposed a new consensus protocol to meet the physical limits of a fixed network. Here, by 'network', we mean the entire networking stack, ranging from the hardware to the topology to the relaying protocols. In Chapter 3, this whole stack is represented with a simple model for worst-case block delay, described in Section 3.3. This model is fairly standard in the consensus literature, but it abstracts away most of the nuances that characterize delays in real networks. In Chapter 4, we consider a more detailed, random network model informed by measurements of real cryptocurrencies. Armed with this more sophisticated network model, we propose a new networking protocol that improves throughput for any consensus algorithm from a broad class of protocols, including Nakamoto consensus and Prism. This protocol, called Barracuda, requires every proposer node to poll its peers for new blocks and update its local view of the blocktree before proposing a new block. This deceptively simple polling procedure gives the proposer more information about the global blocktree, thereby reducing the probability of the proposer's new block causing a fork. In fact, we show that if each proposer polls ℓ nodes before proposing, for small values of ℓ, the end effect is the same as if the entire network had been a factor of ℓ faster in mean block delay. Notably, this benefit comes without any hardware upgrades or other substantial changes to the network. Barracuda is therefore a useful primitive for improving blockchain throughput and latency that operates in parallel to the consensus algorithm improvements from Chapter 3.

Chapter 5: Scalable Storage and Computation

Chapter 3 showed how to approach the physical limits on latency and throughput. In Chapter 5, we focus on approaching physical limits on storage and computation efficiency. A trivial upper bound

is one where storage and computational capacity scale linearly in the number of nodes. Put more simply, doubling the number of nodes should also double the amount of data the system can store and computation it can execute. Current blockchain designs operate far from this fundamental limit; indeed, in existing systems, more user participation actually leads to *lower* storage and computational efficiency. This reverse scalability arises in part because of the global requirements of cryptocurrencies: all (or many) users are expected to store and process the blockchain, so the resulting storage and computational costs increase with time and user participation. Current proposals for improving scalability (called *sharding*) address this tradeoff by improving storage and computational efficiency; however, they typically do so at the cost of security. The more shards exist in a system, the easier it becomes for an adversary to overtake any single shard. As before, a natural question is whether one can simultaneously achieve optimal storage efficiency, computational efficiency, *and* security.

In this chapter, we answer this question affirmatively by presenting PolyShard, a storage and computation solution that grows *more* efficient with more users without sacrificing security. PolyShard combines classical ideas from coded storage and coded computation with recent breakthrough results on Lagrange-coded computing to circumvent the tradeoffs that characterize most sharding systems. The key intuition is that it mixes up data from different users and transactions in a way that allows perfect recovery, illustrated intuitively in Figure 2.3. Notably, PolyShard simultaneously achieves an optimal tradeoff between security, storage, and computational costs, performing within a small constant factor of the limits imposed by physics.

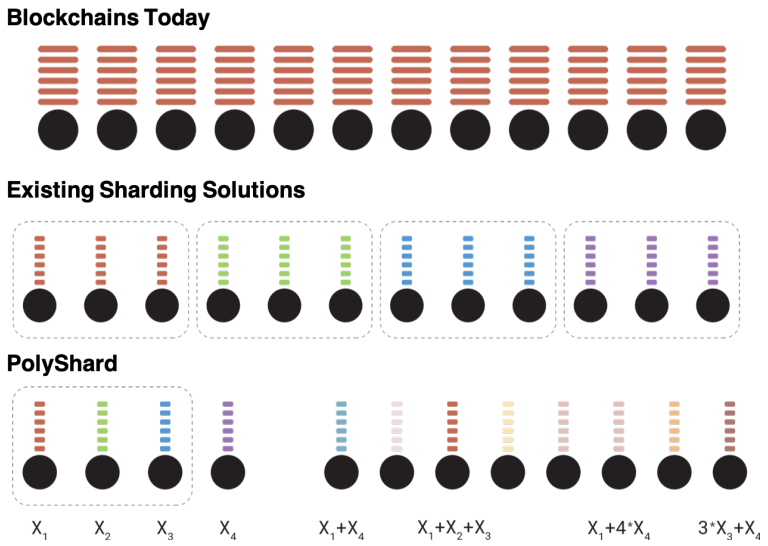


Figure 2.3: PolyShard is a sharding solution that uses ideas from coding theory to simultaneously achieve optimal guarantees in security, storage efficiency, and computational efficiency. The key intuition is that nodes should not store replicated data; instead, they should store *coded* linear combinations of data.

Chapter 6: Payment Channel Networks

In Chapters 3 and 5 we propose layer 1 algorithms that meet physical limits on throughput, latency, storage, computation, and security. These physical limits apply specifically to distributed consensus systems (i.e., layer 1). In some cases, layer 2 scalability solutions are subject to more lenient physical constraints; this can lead to further efficiency gains if harnessed properly.

A notable example is payment channel networks (PCNs). PCNs improve the scalability of

cryptocurrencies by removing the need to involve the blockchain with every transaction; to achieve this, users escrow money on the blockchain ahead of time, and use cryptographic data structures to quickly and efficiently draw from these escrows later. Examples of PCNs include the Lightning network in Bitcoin and the Raiden network in Ethereum. Despite their promise, payment networks presents numerous technical challenges that have yet to receive much attention from the research community. In particular, the routing algorithms used in today's PCNs are naive compared to routing in more mature networks, like the Internet or data centers.

In Chapter 6 we present Spider, a networking solution for PCNs that exploits tried-and-tested ideas from the networking literature. Spider uses packet-switched, imbalance-aware routing to reduce costs and rebalancing frequency within a PCN. In other words, it reduces the amount of money that must be escrowed to support a given transaction load. We find that for stable transaction workflows Spider achieves within a small factor of optimal throughput. In simulation, Spider achieves up to 50% higher transaction throughput compared to state-of-the-art algorithms, without sacrificing transaction processing delay.

Chapter 7: Economics

Economics are a central, open question in the design of cryptocurrencies: what is the value of a cryptocurrency? How should we design reward mechanisms to incentivize rational players to participate in blockchain systems? These questions are typically answered in an ad-hoc manner, motivated primarily by philosophical or qualitative arguments. The goal of Chapter 7 is to provide a framework for answering these questions quantitatively.

A common view is that cryptocurrencies should be valued according to their functionality. So a payment token like Unit-e should be valued according to the fiat value of transactions it processes. However, translating this high-level thought into a precise valuation is challenging; typically, token values are tied to token velocity, which is difficult to pin down. Moreover, existing models have primarily covered proof-of-work cryptocurrencies, where the high hardware and electricity costs materially affect the value of the network. These models do not extend to proof-of-stake tokens, where physical costs are much smaller.

In this chapter, we design quantitative models for valuing proof-of-stake payment tokens based on conventional asset valuation techniques. These models rely on one key observation: in proof-of-stake cryptocurrencies, users deposit tokens to participate in consensus protocols that secure the network. As compensation for participating in these protocols, they receive rewards in the form of transaction fees. Thus, the value of a single deposited token is determined by the cumulative future rewards the owner will receive from the consensus mechanism. By assuming transaction fees are a fixed percentage of transaction value, we can tie the overall token value to the fiat value of transactions executed on the network.

Chapter 8: Privacy and Identity Management

The notion of identity is critical to financial transactions, both to customers and merchants. A key observation of cryptocurrencies is that payment systems do not need to know the identities of their participants; this is similar to cash and barter system economies. In this chapter we explore the fundamental challenges arising from the need to maintain strong identity management between relevant parties (i.e., merchants and customers), while cutting out unnecessary information leakage to the middleman. We study these challenges in two parts: the first part involves identity management among network participants, such as merchants and clients. The second part deals with the technically much harder problem of identity management in the blockchain. This latter aspect

is the specific focus of Chapter 8; we study privacy both at the blockchain level (i.e. preventing transaction leakage) and at the network level (preventing linkage between transactions and IP addresses).

We start the chapter by describing existing techniques for providing blockchain data privacy using zero-knowledge techniques. We summarize primary scalability challenges, and outline the tradeoffs between different approaches, including zk-SNARKS and zk-STARKS. Next, we discuss the landscape of network-level attacks, including a summary of existing attacks and analysis. Finally, we present a proposed solution for network-level privacy called Dandelion. Dandelion prevents network adversaries from linking transactions to IP addresses; intuitively, it achieves this by changing the way that transactions are relayed over the P2P network to an asymmetric spreading pattern, shown in Figure 2.4. We show that Dandelion achieves optimal privacy guarantees at low latency cost, and discuss practical considerations for implementing Dandelion in a low-latency cryptocurrency like Unit-e.

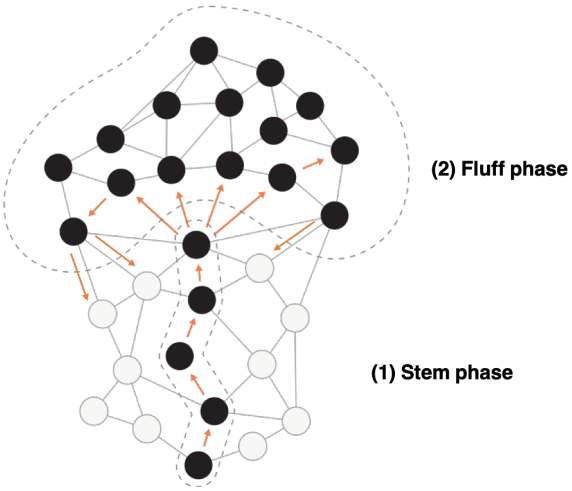


Figure 2.4: Dandelion is a transaction relaying mechanism that prevents network adversaries from linking a transaction to the IP address of the transaction source. It spreads content in two phases: the stem phase passes the transaction along a random walk over an underlying network. The fluff phase diffuses the transaction to the rest of the network.

Chapter 3

Prism: Consensus near Physical Limits

Vivek Bagaria, Stanford
Sreeram Kannan, UW-Seattle
David Tse, Stanford
Giulia Fanti, CMU
Pramod Viswanath, UIUC

3.1 Introduction

In 2008, Satoshi Nakamoto invented the concept of a blockchain, a mechanism to maintain a distributed ledger for an electronic payment system, Bitcoin [130]. Honest nodes mine blocks on top of each other by solving Proof-of-Work (PoW) cryptographic puzzles; by following a longest chain protocol, they can come to consensus on a transaction ledger that is difficult for an adversary to alter. Solving the puzzle effectively involves randomly trying a hash inequality until success. Since Bitcoin's invention, much work has been done on improving Nakamoto's design; however, it remains unclear what is the best performance achievable by blockchain protocols. In this chapter, we explore the performance limits of blockchain protocols and propose a new protocol, Prism, that performs close to those limits. Our focus in this chapter is on Proof-of-Work systems, because the research on these systems are more well-established and hence there are clear baselines to compare our results to. Nevertheless, we will find in the next chapter that many of the ideas are transferrable to designing a Proof-of-Stake version of the protocol.

3.1.1 Performance measures

There are four fundamental performance measures of a PoW blockchain protocol:

1. the fraction β of hashing power the adversary can control without compromising system security;

2. the throughput λ, number of transactions confirmed per second;

3. the confirmation latency, τ, in seconds;

4. the probability ε that a confirmed transaction will be removed from the ledger in the future. ($\log 1/\varepsilon$ is sometimes called the *security parameter* in the literature[1].)

For example, Bitcoin is secure against an adversary holding up to 50% of the total network hash power ($\beta = 0.5$), has throughput λ of the order of several transactions per seconds and confirmation latency of the order of tens of minutes to hours. In fact, there is a tradeoff between the confirmation latency and the confirmation error probability: the smaller the desired confirmation error probability, the longer the needed latency is in Bitcoin. For example, Nakamoto's calculations [130] show that for $\beta = 0.3$, while it takes a latency of 10 blocks (on the average, 100 minutes) to achieve an error probability of 0.04, it takes a latency of 30 blocks (on the average, 300 minutes) to achieve an error probability of 10^{-4}. This latency arises because in order to provide a low error probability, blocks must be deep in the underlying blockchain to prevent the adversary from growing a longer side chain and overwriting the block in question.

3.1.2 Physical limits

Bitcoin has strong security guarantees, being robust against an adversary with up to 50% hashing power. However, its throughput and latency performance are poor; in particular high latency is required to achieve very reliable confirmation. Much effort has been expended to improve the performance in these metrics while retaining the security guarantee of Bitcoin. But what are the fundamental bounds that limit the performance of *any* blockchain protocol?

Blockchains are protocols that run on a distributed set of nodes connected by a physical network. As such, their performance is limited by the attributes of the underlying network. The two most important attributes are C, the communication capacity of the network, and D, the speed-of-light propagation delay across the network. Propagation delay D is measured in seconds and the capacity C is measured in transactions per second in this manuscript, since a transaction is the basic unit of information in a payment system. Nodes participating in a blockchain network need to communicate information with each other to reach consensus; the capacity C and the propagation delay D limit the *rate* and *speed* at which such information can be communicated. These parameters encapsulate the effects of both fundamental network properties (e.g., hardware, topology), as well as resources consumed by the network's relaying mechanism, such as validity checking of transactions or blocks. Assuming that each transaction needs to be communicated at least once across the network, it is clear that λ, the number of transactions which can be confirmed per second, is at most C, i.e.

$$\lambda < C. \tag{3.1}$$

One obvious constraint on the confirmation latency τ is that

$$\tau > D. \tag{3.2}$$

Another less obvious constraint on the confirmation latency comes from the network capacity and the reliability requirement ε. Indeed, if the confirmation latency is τ and the block size is B transactions, then at most

$$\frac{C}{B} \cdot \tau$$

mined blocks can be communicated across the network during the confirmation period for a given transaction. These mined blocks can be interpreted as confirmation *votes* for a particular transaction during this period; i.e. votes are communicated at rate C/B and $\tau \cdot C/B$ votes are accumulated over

[1] All logarithms in this chapter are taken with respect to base e.

duration τ. This number is maximized at $C\tau$, when the block size is smallest possible, i.e. $B = 1$. On average, a fraction $\beta < 0.5$ of these blocks are adversarial, but due to the randomness in the mining process, there is a probability, exponentially small in $C\tau$, that there are more adversarial blocks than honest blocks; if this happens, confirmation cannot be guaranteed. Hence, this probability is a lower bound on the achievable confirmation error probability, i.e.

$$\varepsilon = \exp\{-O(C\tau)\}. \tag{3.3}$$

Turning this equation around, we have the following lower bound on the latency for a given reliability requirement ε:

$$\tau = \Omega\left(\frac{1}{C} \cdot \log\frac{1}{\varepsilon}\right). \tag{3.4}$$

Comparing the two constraints (3.2) and (3.4), we see that if

$$CD \gg \log\frac{1}{\varepsilon},$$

the latency is limited by the propagation delay; otherwise, it is limited by the confirmation reliability requirement. The quantity CD is analogous to the key notion of *bandwidth-delay product* in networking (see eg. [97]); it is the number of "in-flight" transactions in the network.

To evaluate existing blockchain systems with respect to these limits, consider a global network with communication links of capacity 20 Mbits/second and round-the-world speed-of-light propagation delay D of 0.2 seconds. If we take a transaction of size 100 bytes, then $C = 25,000$ transactions per second. The bandwidth-delay product $CD = 5000$ is very large. Hence, the confirmation latency is limited by the propagation delay of 0.2 seconds, but not by the confirmation reliability requirement unless it is astronomically small. Real-world blockchain systems operate far from these physical network limits. Bitcoin, for example, has λ of the order of 10 transactions per second, τ of the order of minutes to hours, and is limited by the confirmation reliability requirement rather than the propagation delay. Ethereum has $\lambda \approx 15$ transactions per second and $\tau \approx 3$ minutes to achieve an error probability of 0.04 for $\beta = 0.3$ [36].

3.1.3 Main contribution

The main contribution of this work is a new blockchain protocol, Prism, which has the following provable performance guarantees:

1. **security:** Prism is secure up to an adversary power of 50%, i.e. for any $\beta < 0.5$ and for arbitrary adversarial action[2], it can achieve an eventual total ordering of the transactions, with consistency and liveness guarantees.

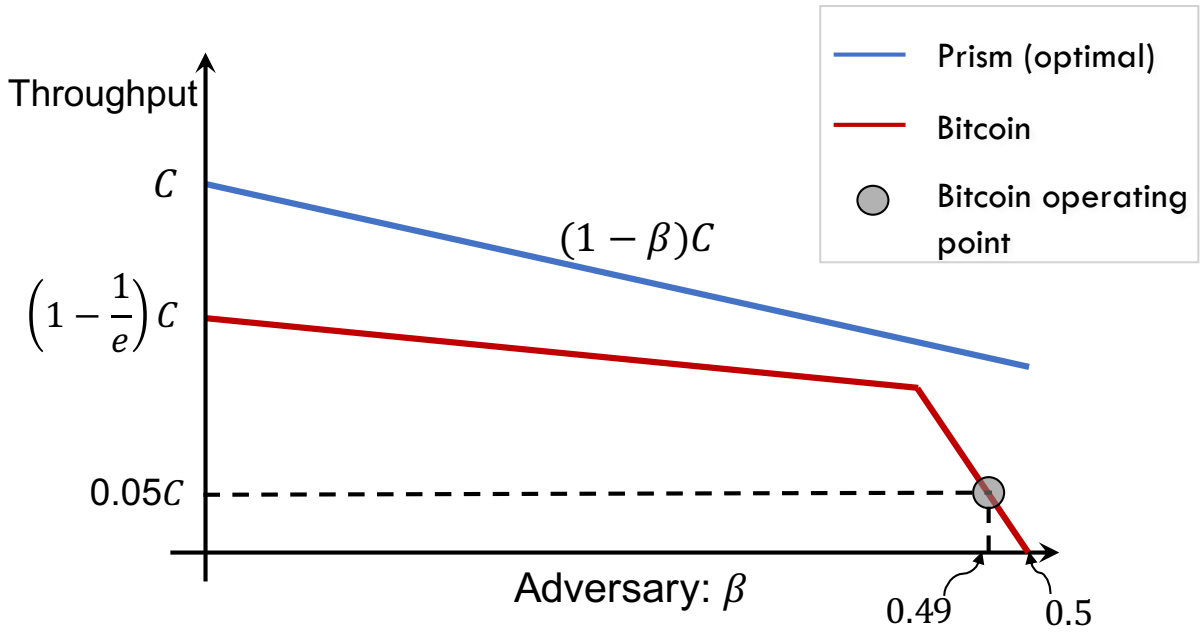
2. **throughput:** For arbitrary adversarial action, Prism can achieve a throughput

$$\lambda = (1-\beta)C \quad \text{transactions per second.}$$

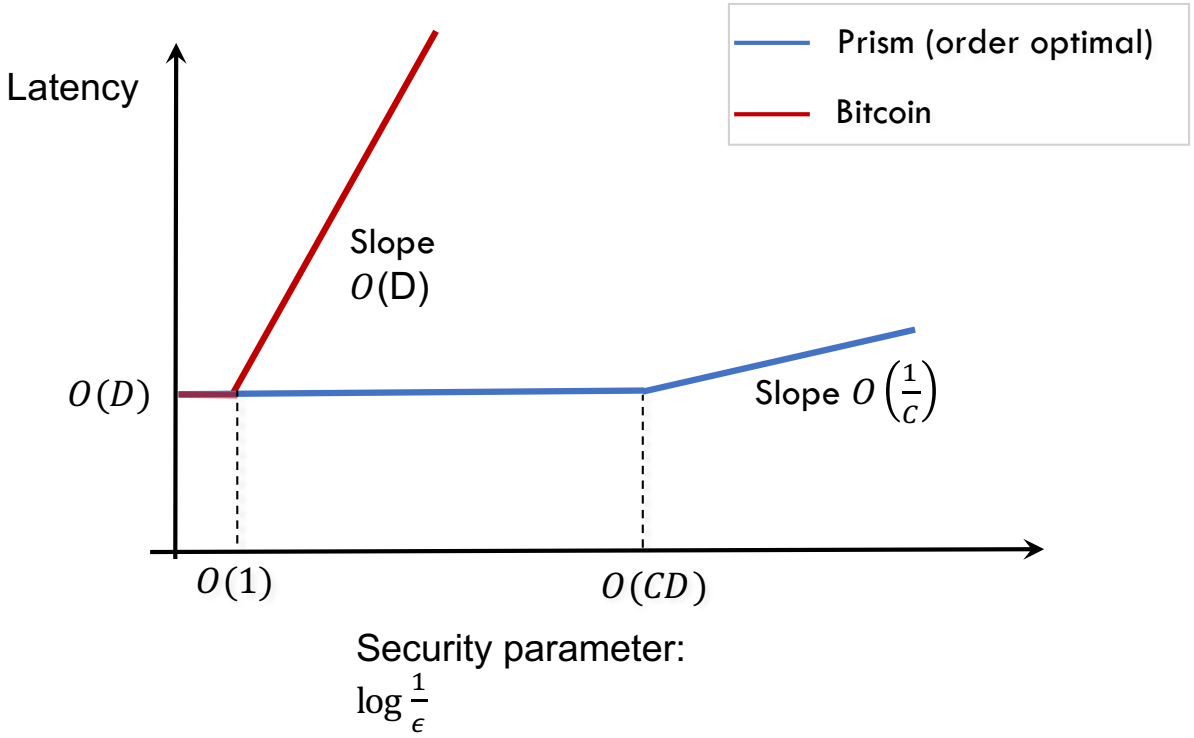
3. **latency:** For any $\beta < 0.5$ and for arbitrary adversarial action, Prism can confirm honest transactions (without public double spends) with an expected latency

$$\mathbb{E}[\tau] < \max\left\{a_1(\beta)D, \frac{a_2(\beta)}{C}\log\frac{1}{\varepsilon}\right\} \quad \text{seconds,}$$

with confirmation reliability at least $1-\varepsilon$. Here, $a_1(\beta)$ and $a_2(\beta)$ are constants depending only on β (defined in (3.28) and (3.29)).



(a) Throughput



(b) Latency

Figure 3.1: Main results. (a) Throughput vs adversarial fraction β for Prism and Bitcoin. The red curve is an optimized upper bound of Bitcoin's throughput, derived in Section 3.4.1. Note that the throughput of Prism is a positive fraction of the network capacity all the way up to $\beta = 0.5$, but the throughput of Bitcoin vanishes as a fraction of the capacity as $\beta \to 0.5$. (b) Confirmation latency vs. security parameter for Prism and Bitcoin. The red curve is a lower bound on Bitcoin 's latency, derived in Section 3.5.1. The latency of Prism is independent of the security parameter value up to order CD and increases very slowly after that (with slope $1/C$). For Bitcoin, latency increases much more rapidly with the security parameter.

The results are summarized in Figure 3.1. Some comments:

- The security of Prism is as good as Bitcoin: Prism can be robust to an adversary with hashing power up to $\beta = 0.5$.
- Since $1 - \beta$ is the fraction of honest hashing power, Prism 's throughput is optimal assuming each transaction needs to be communicated across the network.
- Prism achieves a confirmation latency for honest transactions matching, in order, to the two physical limits (3.2) and (3.4). In particular, if the desired security parameter $\log \frac{1}{\varepsilon} \ll CD$, the confirmation latency is of the order of the propagation delay and *independent* of $\log 1/\varepsilon$. Put it another way, one can achieve latency close to propagation delay with a confirmation error probability exponentially small in the bandwidth-delay product CD.
- For a total ordering of all transactions (including double spends), on the other hand, the trade off between latency and the security parameter is similar to that of Bitcoin.

3.1.4 Approach

A critical parameter of any PoW blockchain protocol is the mining rate, i.e. the rate at which puzzles are successfully solved (also called the PoW solution rate). The mining rate can be easily controlled via adjusting the difficulty of the puzzle, i.e. the threshold at which the hash inequality needs to be satisfied. The mining rate has a profound impact on both the transaction throughput and confirmation latency. Large mining rate can potentially increase the transaction throughput by allowing transactions to be processed quicker, and can potentially reduce the confirmation latency by increasing the rate at which votes are casted to confirm a particular transaction. However, increasing the mining rate has the effect of increasing the amount of forking in the blocktree, because blocks mined by different nodes within the network delay cannot be mined on top of each other and are hence forked. This de-synchronization slows down the growth rate of the longest chain, making the system more vulnerable to private chain attacks, and decreasing the security of the protocol. Indeed, one reason why Bitcoin is highly secure is that the mining rate is set to be very small, one block per 10 minutes. At the current Bitcoin block size of 1 Mbytes, this corresponds to a generated traffic of about 13 kbits/second, much less than capacity of typical communication links [170]. Thus, Bitcoin 's performance is security-limited, not communication-limited, and far away from the physical limits.

To increase the mining rate while maintaining security, one line of work in the literature has used more complex fork choice rules and/or added reference links to convert the blocktree into more complex structures such as a directed acyclic graph (DAG). This allows a block to be confirmed by other blocks that are not necessarily its descendents on a main chain. (Figure 3.2). Examples of such works are GHOST [168], Inclusive [112], Spectre [167], Phantom[166] and Conflux [114]. However, as discussed in more details in the related work section, GHOST, Phantom, and Conflux all have security issues, and Spectre does not provide total ordering of transactions. It is fair to say that handling a highly forked blocktree is challenging.

In this work, we take a different approach. We start by *deconstructing* the basic blockchain structure into its atomic functionalities, illustrated in Figure 3.3. The selection of a main chain in a blockchain protocol (e.g., the longest chain in Bitcoin) can be viewed as electing a leader block among all the blocks at each level of the blocktree, where the level of a block is defined as its distance (in number of blocks) from the genesis block. Blocks in a blockchain then serve three purposes: they elect leaders, they add transactions to the main chain, and they vote for ancestor blocks through

[2]The precise class of allowable adversarial actions will be defined in the formal model.

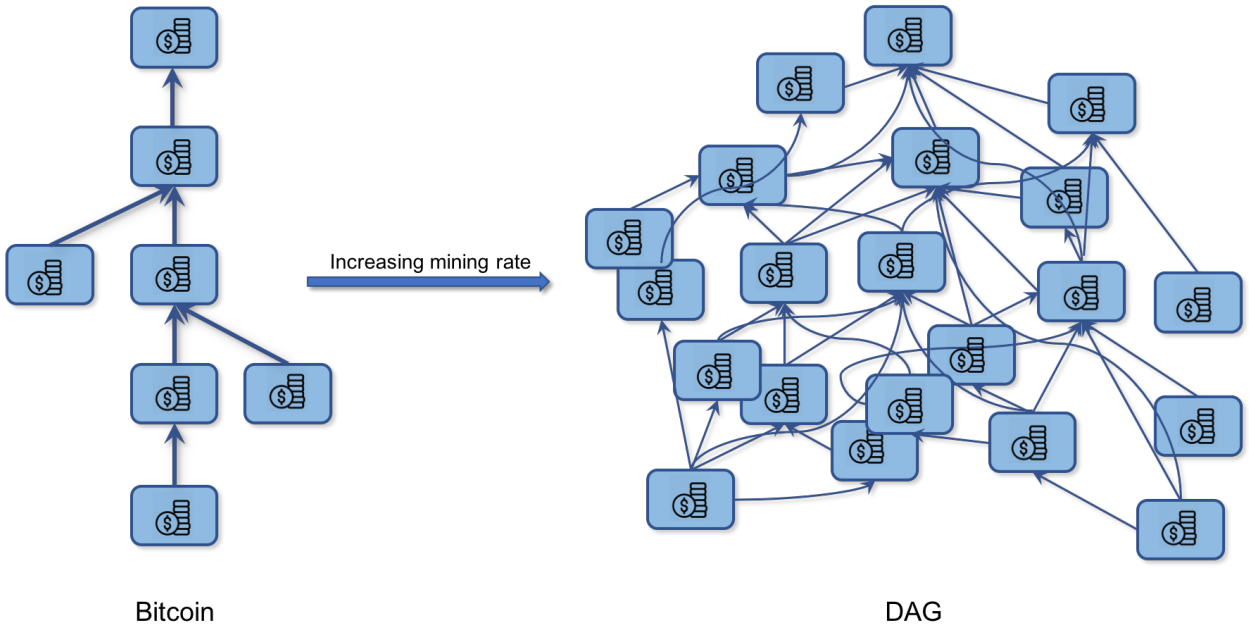


Figure 3.2: The DAG approach to increasing the mining rate.

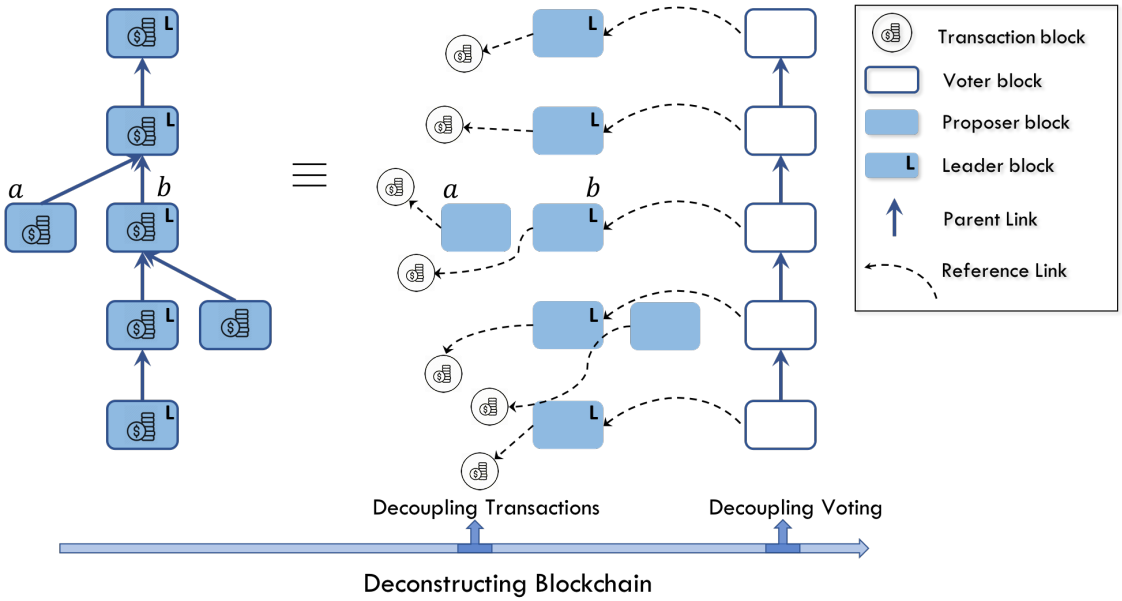


Figure 3.3: Deconstructing the blockchain into transaction blocks, partially ordered proposal blocks arranged by level, and voter blocks organized in a voter tree. The main chain is selected through voter blocks, which vote among the proposal blocks at each level to select a leader block. For example, at level 3, block b is elected the leader over block a.

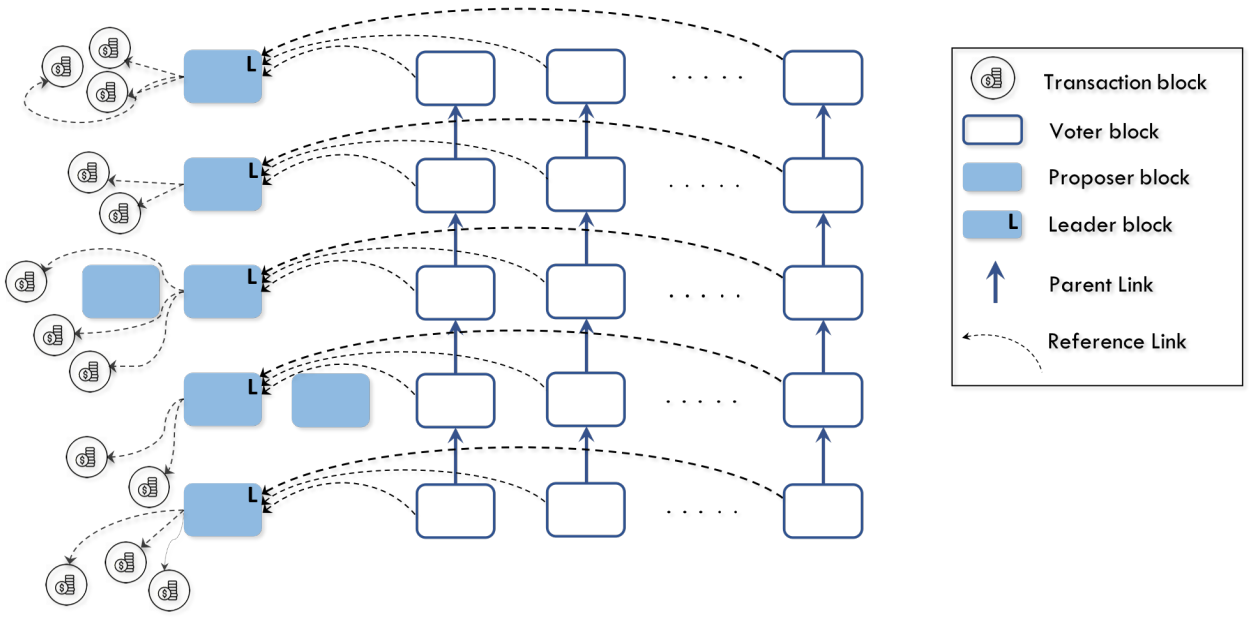


Figure 3.4: Prism. Throughput, latency and reliability are scaled to the physical limits by increasing the number of transaction blocks and the number of parallel voting chains per proposal block.

parent link relationships. We explicitly separate these three functionalities by representing the blocktree in a conceptually equivalent form. In this representation, blocks are divided into three types: proposer blocks, transaction blocks and voter blocks. The voter blocks vote for transactions indirectly by voting for proposer blocks, which in turn link to transaction blocks. Proposer blocks are grouped according to their level in the original blocktree, and each voter blocktree votes among the proposer blocks at the same level to select a leader block among them. The elected leader blocks can then bring in the transactions to form the final ledger. The voter blocks are organized in their own blocktree and support each other through parent links. Thus, the parent links in the original blocktree have two implicit functionalities which are explicitly separated in this representation: 1) they provide a partial ordering of the proposal blocks according to their levels, and 2) they help the voting blocks to vote for each other.

This alternative representation of the traditional blockchain, although seemingly more complex than the original blockchain representation, provides a natural path for scaling the performance of blockchain protocols to approach physical limits (Figure 3.4). To increase the transaction throughput, one can simply increase the number of transaction blocks that a proposer block points to without compromising the security of the blockchain. This number is limited only by the physical capacity of the underlying communication network. To provide fast confirmation, one can increase the number of parallel voting trees, with many voters voting on the proposal blocks in parallel, until reaching the physical limit of confirming with speed-of-light latency and extremely high reliability. Note that even though the overall block generation rate has increased tremendously, the number of proposal blocks per level remains small and manageable, and the voting blocks are organized into many separate voting chains with low block mining rate per chain and hence little forking. The overall structure, comprising of the three kinds of blocks and the links between them, is a DAG, but a structured DAG.

This complexity management presupposes a way to provide *sortition* in the mining process: when miners mine for blocks, they should not know in advance whether the block will become a proposal

block, a transaction block, or a voting block, and if it is a voting block, it should not know in advance what particular chain the voting block will be in. Otherwise an adversary can focus its hashing power to attack a particular part of the structure. This sortition can be accomplished by using the random hash value when a block is successfully mined; this is similar to the 2-for-1 PoW technique used in [79], which is also used in Fruitchains [137] for the purpose of providing fairness in rewards. In fact, the principle of *decoupling* functionalities of the blockchain, central to our approach, has already been applied in Fruitchains, as well as other works such as BitcoinNG. This line of work will be discussed in depth in Section 3.2, but its focus is only on decoupling the transactions-carrying functionality. In our work, we broaden this principle to decouple *all* functionalities.

In Bitcoin, the irreversibility of a block in the longest chain is achieved by a *law of large numbers* effect: the chance that an adversary with less that 50% hashing power can grow a private chain without the block and longer than the public chain diminishes with the depth of the block in the public chain. This is the essence of the random walk analysis in Nakamoto's original paper [130] and is also implicit in the formal security analysis of Bitcoin in [79] (through the definition of *typical execution*). The law of large numbers allows the averaging of the randomness of the mining process, so that the chance of the adversary getting lucky and mining many blocks in quick succession is small. This averaging is achieved over time, and comes at the expense of long latency, which increases with the desired level of reliability.

Prism also exploits the law of large numbers, but over the number of parallel voter trees instead of over time. Due to the sortition mechanism, the mining processes of both the adversary and the honest nodes are independent across the voting trees. By having many such trees, many votes are cast on the proposer blocks at a given level, and the chance of an adversary with less than 50% hashing power being able to reverse many of these votes decreases exponentially with m, the number of voter trees. The number of voter trees is m, and hence the rate of vote generation, is limited only by the physical capacity C of the network. Thus, we can attain irreversibility of a large fraction of the votes with high probability (approaching 1 exponentially fast in the bandwidth-delay product CD) without waiting for a long time. We show that this irreversibility of votes allows fast confirmation of a final leader block among a list of proposer blocks at a given level. In particular, it is guaranteed that the adversary cannot propose another block in the future that has enough votes to become the final leader block at that level. The ability to do this for all levels up to a given level generates a list of transaction ledgers, one of which must be a prefix of the eventual totally-ordered ledger (Figure 3.5). Together with liveness of honest transactions, we show that this "list decoding" capability is sufficient for fast confirmation of all honest transactions[3]. If a given block obtains a substantial enough majority of votes, then the list can be narrowed to contain only that block, which can then be declared the leader block. In the worst case, when votes are tied between two or more proposer blocks at the same level (due to active intervention by the adversary, for example), the irreversibility of *all* of the votes and a content-dependent tie-breaking rule is needed to come to global consensus on a unique leader block; this requires higher latency. Hence, Prism requires high latency in the worst case to guarantee total ordering of all transactions.

The above discussion gives some intuition behind Prism, but a formal analysis is needed to rigorously establish security, latency and throughput performance guarantees. Such a formal analysis was done on Bitcoin in [79] in a synchronous round-by-round model and subsequently extended in [136] to an asynchronous model with an upper bound on block network delay. In particular, [79] pioneered the backbone protocol analysis framework where it was shown that two key properties, the common-prefix property and the chain-quality property, of the Bitcoin backbone guarantee

[3]List decoding is a concept in coding theory. Instead of decoding to a unique codeword, list decoding generates a list of possible codewords which is guaranteed to contain the true codeword.

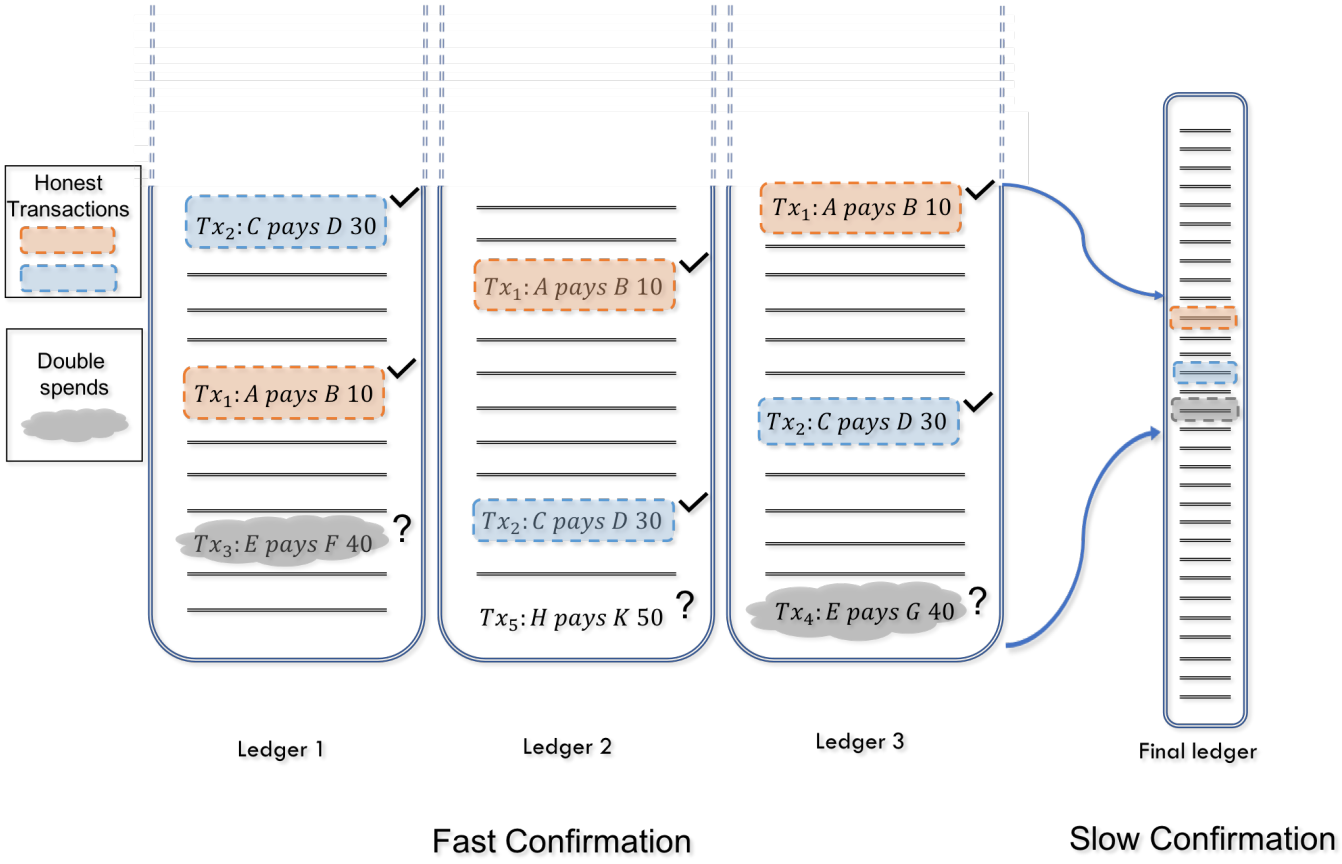


Figure 3.5: List ledger confirmation. An example where one can fast-confirm that the final ledger is one of three possibilities. Honest transactions that appear in all three ledgers can be fast-confirmed. Double spends cannot appear in all ledgers and are therefore not fast-confirmed, although one of them will be slow-confirmed.

persistence and liveness of the ledger maintained by Bitcoin respectively. We leverage this framework to provide a formal analysis of Prism in the synchronous round-by-round model (we conjecture that similar results can be established in the more sophisticated asynchronous model of [136]). Technically, the most challenging part of the analysis is on fast latency confirmation, where we show that: 1) the common-prefix property of the vote trees guarantees *vote persistence*, so that a large fraction of the votes will not be reversed; 2) the chain-quality of the main chains of the vote trees guarantees vote liveness, so that a large fraction of the vote trees will contain honest votes on the proposer blocks at each level of the proposal tree.

3.1.5 Outline of paper

In Section 3.2, we discuss other lines of work in relation to our approach. In Section 3.3, we review the synchronous model used in [79] and introduce our network model that ties the blockchain parameters to physical parameters of the underlying network. In Section 3.4, we focus on throughput, and discuss a simplified version of the protocol, Prism 1.0, which achieves full security and optimal throughput. Since Prism 1.0 lacks voter blocktrees, it has latency equivalent to Bitcoin. In Section 3.5, we add vote trees to the protocol, and perform a formal analysis of its security and fast latency. The result is a protocol, Prism, which can achieve full security, optimal throughput and near physical limit latency on ledger list decoding and confirmation of honest transactions. In Section 3.6, we will discuss the issue of incentivization, as well as applications of our results to Proof-of-Stake and smart contracts systems.

3.2 Related work

In this section, we discuss and compare our approach to several lines of work.

3.2.1 High-forking protocols

As discussed in the introduction, one approach for increasing throughput and decreasing latency is the use of more sophisticated fork choice and voting rules to deal with the high-forking nature of the blocktree. Examples of such high-forking protocols include GHOST [168], Inclusive [112], Spectre [167], Phantom [166], and Conflux [114]. The earliest of these schemes, GHOST, handles forking through a fork-choice rule that builds on the heaviest subtree [168]. The authors observed that in order to improve throughput, we must increase the block mining rate, f. However, as f grows, so too does the number of blocks mined in parallel, which are wasted under Bitcoin's longest-chain fork choice rule, thereby reducing security. GHOST's heaviest-subtree rule allows the system to benefit from blocks that were mined in parallel by honest nodes since such blocks are counted in the main chain calculation. While it was shown in [168] that GHOST is secured against a 50% purely private attack, it turns out that GHOST is vulnerable to a public-private balancing attack [132], where the adversary can use minimal effort to split the work of the honest nodes across two subtrees of equal weight, and then launch a private attack. It turns out that counting side-chain blocks in selecting the main chain allows the adversary to withhold earlier mined blocks and use them at later times to balance the growth of the two subtrees. We present an analysis of this attack in the Appendix and show that this attack restricts the mining rate f of GHOST to be similar to that of Bitcoin, thus minimizing the advantage of GHOST.

To improve security at high mining rates, another popular idea is to add reference links between blocks in addition to traditional parent links, resulting in a DAG-structured blockchain. Each block in a DAG can reference multiple previous blocks instead of a unique ancestor (as in Bitcoin). The

pertinent challenges are how to choose the reference links and how to obtain a total ordering of blocks from the observed DAG in a way that is secure. In a family of protocols, Inclusive, Spectre and Phantom, every block references all previous orphan blocks. These reference links are interpreted in differing ways to give these different protocols. For example, in [112], the key observation is that the reference link structure provides enough information to emulate any main-chain protocol, such as the longest-chain or GHOST protocol, while in addition providing the ability to pull in stale blocks into a valid ledger. However, the security guarantee remains the same as that of Bitcoin (namely, tending to zero as the mining rate grows), and it does not achieve optimal throughput.

Spectre is an innovative scheme that builds upon the the DAG idea to achieve low confirmation time by interpreting the reference links as votes to compare between pairs of blocks [167]. However, the fast confirmation is restricted to honest transactions and the system does not guarantee liveness for double-spends as well as not having the ability to confirm smart contracts that need a totally-ordered ledger. Since complete ordering is important for core blockchain applications (e.g., cryptocurrencies), a later work, Phantom, builds on Spectre to achieve consensus on a total ordering of transactions by having participants topologically sort their local DAGs [166]. The authors suggest that by combining Spectre and Phantom, one may be able to achieve low confirmation latency for honest transactions as well as eventual total ordering. However, a recent work [114] demonstrates a liveness attack on Phantom. Furthermore, the proposed hybrid scheme cannot confirm non-contentious smart contracts with fast latency. Although Prism uses a DAG to order transactions, it diverges from prior DAG schemes by separating block proposal from block ordering in the protocol. This helps because an adversarial party that misbehaves during block proposal does not affect the security of transaction ordering, and vice versa; it provides a degree of algorithmic sandboxing.

Conflux is another DAG-based protocol whose goal is to increase throughput [114]. However, Conflux's reference links are not used to determine where to mine blocks or how to confirm them; they are only used to include side-chain blocks into the main chain to improve throughput. The main chain itself is selected by the GHOST rule. Due to the vulnerability of GHOST to the balancing attack, the secured throughput of Conflux is limited to Bitcoin levels. (See discussions in Section 3.4.6.)

3.2.2 Decoupled consensus

Our design approach is based on the principle of *decoupling* the various functionalities of the blockchain. This decoupling principle has already been applied in various earlier works, but mainly in decoupling the transactions. We review these works here.

BitcoinNG [63] elects a single leader to propose a predetermined number of transaction blocks, called an epoch. At the end of this epoch, a new leader is elected. Thus, there is a decoupling of proposal blocks and transaction blocks, the goal being to increase the throughput. BitcoinNG is vulnerable to various security attacks, such as bribery or DDoS attacks, whereby an adversary can corrupt a block proposer after learning its identity. In contrast, Prism does not reveal the identity of a block proposer *a priori*.

In BitcoinNG, if the leader is malicious, it may try to double spend transaction blocks during its epoch. Although this event is handled by the BitcoinNG confirmation rule, which only confirms transactions once they are deep enough in the chain of keyblocks, the confirmation rule incurs a latency comparable to Bitcoin's. ByzCoin [104] and its predecessor DiscCoin [57] address this point by using Byzantine fault-tolerant protocols in an architecture similar to BitcoinNG's. Like BitcoinNG, ByzCoin separates time into epochs, where each epoch is associated with a new leader and validation committee; these are determined by the blockchain. In a given epoch, the leader proposes frequent transaction blocks, which the committee commits (and signs). In parallel, a separate chain of

infrequent, PoW keyblocks mark the end of each epoch, and the composition of the next committee. Transaction blocks contain pointers to both the most recent transaction block and keyblock; all blocks are collectively signed [171], which prevents forking among transaction blocks. Notably, ByzCoin transaction blocks are not mined but are put on the chain by the committee leader, and confirmed by a fixed committee. Hence, the leader and/or committee nodes remain subject to DDoS or bribery attacks.

The objective of Fruitchains [137] is to provide better chain quality compared to Bitcoin; at a high level, chain quality refers to the fraction of blocks in the main chain belonging to the adversary. In Bitcoin, adversaries can augment this fraction relative to their computational power by using strategic mining and block release policies, such as selfish mining [64, 156, 133]. Fruitchains mechanically resembles Nakamoto consensus, except miners now mine separate mini-blocks, called fruits, for each transaction. Fairness is achieved because the fraction of fruits a miner can mine is proportional to its computational power. As in BitcoinNG , the fruits (transactions) are decoupled from the proposal blocks in the blocktree, but for a different reason: to improve fairness.

3.2.3 Hybrid blockchain-BFT consensus

Another line of work to improve throughput and latency combines ideas from Byzantine fault tolerant (BFT) along with blockchains. *Hybrid consensus* uses a combination of traditional mining under a longest-chain fork choice rule with Byzantine fault tolerant (BFT) consensus [138]. The basic idea is that every k blocks, a BFT protocol is run by an elected committee of nodes. Hybrid consensus is designed to provide *responsiveness*, which describes systems whose performance depends on the actual network performance rather than an *a priori* bound on network delays. The authors show that no responsive protocol can be secure against more than 1/3 adversarial power, and hybrid consensus achieves this bound. In this work, our focus is not on being responsive to network delay, but close to the propagation delay physical limit and small error probability.

A closely-related protocol called Thunderella includes a slow Nakamoto consensus blockchain, as well as a faster confirmation chain that is coordinated by a leader and verified by a BFT voting protocol [140]. Thunderella achieves low latency under optimistic conditions, including having a honest leader and a $\beta < 0.25$, while having consistency under worst case condition ($\beta = 0.5$). In contrast, our protocol achieves low latency under all conditions, but for list-decoding and confirmation of honest transactions.

Along similar lines, ByzCoin separates time into epochs, where each epoch is associated with a new leader and validation committee; these are determined by the blockchain. Each epoch runs a BFT-style algorithm where the leader proposes frequent transaction blocks, which are signed by the committee. In parallel, a separate chain of infrequent, PoW keyblocks mark the end of each epoch, and the composition of the next committee. Transaction blocks contain pointers to both the most recent transaction block and keyblock; all blocks are collectively signed [171], which prevents forking among transaction blocks. Notably, ByzCoin transaction blocks are not mined but are put on the chain by the committee leader, and confirmed by a fixed committee. Hence, the leader and/or committee nodes remain subject to DDoS or bribery attacks.

3.3 Model

3.3.1 Mining and communication model

Let $\mathcal{N}$ denote the set of participating nodes in the network. Each transaction is a cryptographically secure payment message. When a transaction arrives at the network, it is assumed to be

instantaneously broadcast to all nodes in the network. A *block* consists of an ordered list of B transactions and a few reference links to other blocks. Each node $n \in \mathcal{N}$ controls p_n fraction of total hashing power and it creates blocks from the transactions and mines them with Poisson process rate fp_n blocks per second. There are two types of nodes – honest nodes, $\mathcal{H} \subset \mathcal{N}$, who strictly follow the protocol, and the adversarial nodes, $\mathcal{N}/\mathcal{H}$, who are allowed to not follow the protocol. The adversarial nodes control β fraction of hashing power i.e, $\sum_{v \in \mathcal{N}/\mathcal{H}} p_v = \beta$, whereas the honest nodes control the other $1-\beta$ fraction of hashing power[4]. As a consequence, the honest nodes mine blocks with Poisson process rate $\sum_{v \in \mathcal{H}} fp_v = (1-\beta)f$ and the adversarial nodes mine blocks with Poisson process rate $\sum_{v \in \mathcal{N}/\mathcal{H}} fp_v = \beta f$. Without loss of generality we can assume a single adversarial node with β fraction of hashing power.

The nodes exchange blocks via a broadcast channel. The time taken transmitting a block from one honest node to another honest node is assumed to be Δ seconds. On the other hand, the adversary can transmit and receive blocks with arbitrary delay, up to delay Δ.

To simplify our analysis, we discretize the above continuous-time model into the discrete-time round-by-round synchronous model proposed in [79]. Each round in this model corresponds to Δ seconds in the continuous-time model above. In the rth round, let $H[r]$ and $Z[r]$ be the number of blocks mined by the honest nodes and by the adversarial nodes respectively. The random variables $H[r]$ and $Z[r]$ are Poisson distributed with means $(1-\beta)f\Delta$ and $\beta f\Delta$ respectively and are independent of each other and independent across rounds. The $H[r]$ blocks are broadcast to all the nodes during the round, while the adversary may choose to keep some or all of the $Z[r]$ blocks in private. The adversary may also choose to broadcast any private block it mined from previous rounds. The adversary is allowed to first observe $H[r]$ and then take its action for that round. At the end of each round, all nodes in the network have a common view of the public blocktree.

An important random variable is $Y[r]$, which equals 1 when $H[r] = 1$ and 0 otherwise. This is the indicator random variable for whether the rth round is a *uniquely successful* round, i.e. a round in which only one block is mined by the honest nodes [79]. Note that $Y[r]$ has a Bernoulli distribution with parameter $(1-\beta)f\Delta e^{-(1-\beta)f\Delta}$. Another important random variable is $X[r]$, which equals 1 when $H[r] \geq 1$. We denote $H[r_1, r_2] := \sum_{r=r_1+1}^{r_2} H[r]$, similarly for X, Y and Z.

The location of the $H[r]$ honest blocks in the block data structure after the rth round is protocol-dependent. In Bitcoin, for example, all honest blocks are appended to the longest chain of the public blocktree from the previous round. Adversarial blocks can instead be mined on any public or private block from the previous round.

Following the Bitcoin backbone protocol model [79], we consider protocols that execute for a finite number of rounds, $r_{\max}$ which we call the execution horizon. We note that we do not consider cryptographic failure events, such as insertion in the blockchain, since it has been demonstrated already in the backbone protocol paper that for a polynomial number of rounds $r_{\max}$ in the hash-length, these events have negligible probability.

3.3.2 Network model

To connect to the physical parameters of the network, we assume a very simple network model. The network delay Δ is given by:

$$\Delta = \frac{B}{C} + D, \tag{3.5}$$

i.e. there is a processing delay of B/C followed by a propagation delay of D seconds. This is the same model used in [168], based on empirical data in [56], as well in [149]. However, here, we put an

[4] β for **b**ad.

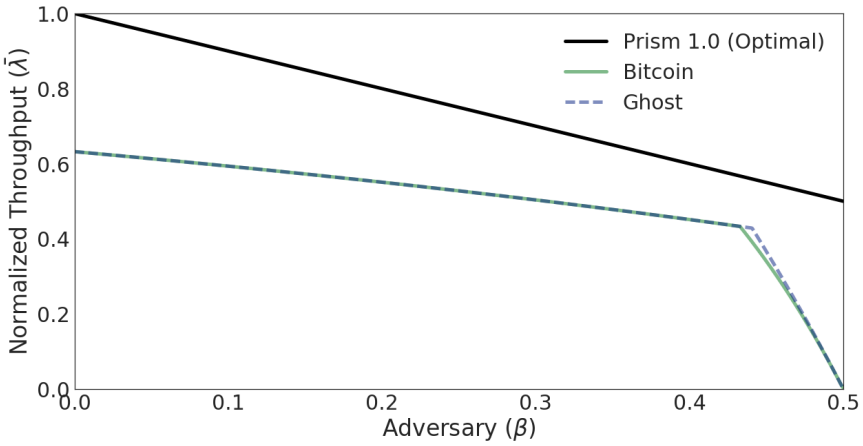


Figure 3.6: Throughput efficiency versus β tradeoff of baseline protocols and Prism 1.0 . The tradeoffs for the baseline protocols are upper bounds, while that for Prism 1.0 is exact.

additional qualification: this expression is valid only assuming the network is stable, i.e. the total workload of communicating the blocks is less than the network capacity. In terms of our parameters:

$$fB < C. \tag{3.6}$$

For a given block size, (3.6) imposes a physical constraint on the total mining rate f. This *stability constraint* sets our model apart from prior work, which has traditionally assumed infinite network capacity; in particular, this gives us a foothold for quantifying physical limits on throughput and latency.

Note that the protocols discussed in this manuscript can be used in any network setting. This simple network model is only used as a common baseline to evaluate how well a particular protocol performs relative to the physical limits. In particular, the delay model (3.5) ignores queuing delay due to randomness of the times of the block transmission across the network.

3.4 Approaching physical limits: throughput

In this section, we study the optimal throughput λ achievable under worst-case adversarial behavior for a given adversarial power β. The main results are summarized in Figure 3.6, which show plots of $\bar{\lambda} := \lambda/C$ versus β for various protocols. The metric $\bar{\lambda}$ is the throughput as a fraction of the network capacity and is a measure of the efficiency of a protocol. The plot shows upper bounds on the efficiency of two baseline blockchain protocols, Bitcoin and GHOST (a version of GHOST is used in Ethereum). Note that the throughput efficiency of both protocols vanishes as β approaches 0.5. In contrast, we design a protocol, Prism 1.0, which attains a throughput efficiency of $1 - \beta$. This efficiency does not vanish as β approaches 0.5 and is in fact the best possible if only honest nodes are working. We will see that the difference between Prism 1.0 and the two baseline protocols is that while the throughput of the two baseline protocols are *security-limited* for large β, the throughput of Prism 1.0 is only limited by the physical network capacity for *all* $\beta < 0.5$.

3.4.1 Baselines: Bitcoin and GHOST

We derive upper bounds on the achievable throughput under worst-case adversarial behavior of two key baselines: Bitcoin and GHOST. Throughput can be altered by tuning two parameters: the

mining rate f and block size B. We are interested in the maximum achievable throughput efficiency ($\bar{\lambda} := \frac{\lambda}{C}$), optimized over B and f. To simplify notation, we suppress the dependence of $\bar{\lambda}$ on β.

3.4.1.1 Bitcoin

The security and consensus properties of Bitcoin have been studied by Nakamoto [130], and formally by [79] in the synchronous model, followed by the analysis of [136] in the asynchronous model. These works and others (e.g., [168, 102]) show that choice of f and B in Nakamoto consensus has tradeoffs. As the mining rate f grows, forking increases and the maximum tolerable adversarial fraction β shrinks, Similarly, as the block size B grows, the network delay Δ also grows, which causes forking.

An upper bound on the worst case throughput (worst case over all adversary actions) is the rate at which the longest chain grows when no adversary nodes mine. The longest chain grows by one block in a round exactly when at least one honest block is mined. Hence the rate of growth is simply $\mathbb{P}(H(r) > 0)$, i.e.

$$1 - e^{-(1-\beta)f\Delta} \text{ blocks per round,} \tag{3.7}$$

Notice that (3.7) is monotonically increasing in f; hence to maximize throughput, we should choose as high a mining rate as possible.

However, we are simultaneously constrained by security. For Bitcoin's security, [79] shows that the main chain must grow faster in expectation than any adversarial chain, which can grow at rates up to $\beta f\Delta$ in expectation. Hence we have the following (necessary) condition for security:

$$1 - e^{-(1-\beta)f\Delta} > \beta f\Delta. \tag{3.8}$$

Equation (3.8) gives the following upper bound on $f\Delta$, the mining rate per round:

$$f\Delta < \bar{f}_{\text{BTC}}(\beta),$$

where $\bar{f}_{\text{BTC}}(\beta)$ is the unique solution to the equation:

$$1 - e^{-(1-\beta)\bar{f}} = \beta\bar{f}. \tag{3.9}$$

This yields an upper bound on the throughput, in transactions per second, achieved by Bitcoin as:

$$\begin{aligned} \lambda_{\text{BTC}} &\leq [1 - e^{-(1-\beta)\bar{f}_{\text{BTC}}(\beta)}]B/\Delta \\ &= \beta\bar{f}_{\text{BTC}}(\beta)B/\Delta, \end{aligned} \tag{3.10}$$

where the last equality follows from (3.9). Substituting in $\Delta = B/C + D$ and optimizing for B, we get the following upper bound on the maximum efficiency of Bitcoin :

$$\bar{\lambda}_{\text{BTC}} \leq \beta\bar{f}_{\text{BTC}}(\beta),$$

achieved when $B \gg CD$ and $\Delta \gg D$.

Another upper bound on the throughput is obtained by setting f at the capacity limit: $f = C/B$ (cf. (3.6)). Substituting into (3.7) and optimizing over B, this yields

$$\bar{\lambda}_{\text{BTC}} \leq 1 - e^{\beta-1},$$

achieved when $f\Delta = 1$, $B \gg CD$ and $\Delta \gg D$.

Combining the above two bounds, we get:

$$\bar{\lambda}_{\text{BTC}} \leq \min\left\{\beta\bar{f}_{\text{BTC}}(\beta), 1 - e^{\beta-1}\right\}$$

This is plotted in Figure 3.6. Note that for large values of β, the first upper bound is tighter; this is a *security-limited* regime, in which the throughput efficiency goes to zero as $\beta \to 0.5$. This is a manifestation of the (well-known) fact that to get a high degree of security, i.e. to tolerate β close to 0.5, the mining rate of Bitcoin must be small, resulting in a low throughput. Bitcoin currently operates in this regime, with the mining rate one block per 10 minutes; assuming a network delay of 1 minute, this corresponds to a tolerable β value of 0.49 in our model.

For smaller β, the second upper bound is tighter, i.e. this is the *communication-limited* regime. The crossover point is the value of β such that

$$1 - e^{\beta - 1} = \beta,$$

i.e., $\beta \approx 0.43$.

3.4.1.2 GHOST

The GHOST [168] protocol uses a different fork choice rule, which uses the heaviest-weight subtree (where weight is defined as the number of blocks in the subtree), to select the main chain. To analyze the throughput of GHOST, we first observe that when there are no adversarial nodes working, the growth rate of the main chain of GHOST is upper bounded by the growth rate of the main chain under the longest chain rule. Hence, the worst-case throughput of GHOST, worst-case over all adversary actions, is bounded by that of Bitcoin, i.e.

$$1 - e^{-(1-\beta)f\Delta} \quad \text{blocks per round,} \tag{3.11}$$

(cf. (3.7)). Notice that once again, this bound is monotonically increasing in f and we would like to set f largest possible subject to security and network stability constraints. The latter constraint gives the same upper bound as (3.4.1.1) for Bitcoin:

$$\bar{\lambda}_{\mathsf{GHOST}} \leq 1 - e^{\beta - 1}. \tag{3.12}$$

We now consider the security constraint on f. Whereas our security condition for Bitcoin throughput was determined by a Nakamoto private attack (in which the adversary builds a longer chain than the honest party), a more severe attack for GHOST is a balancing attack, analyzed in Appendix A. As shown in that analysis, the balancing attack implies that a necessary condition on f for robustness against an adversary with power β is given by:

$$\mathbb{E}[|H_1[r] - H_2[r]|] > \beta f \Delta, \tag{3.13}$$

where $H_1[r], H_2[r]$ are two independent Poisson random variables each with mean $(1-\beta)f\Delta/2$. Repeating the same analysis as we did for Bitcoin, we get the following upper bound on the maximum efficiency of GHOST:

$$\bar{\lambda}_{\mathsf{GHOST}} \leq \beta \bar{f}_{\mathsf{GHOST}}(\beta), \tag{3.14}$$

where $\bar{f}_{\mathsf{GHOST}}(\beta)$ is the value of $f\Delta$ such that (3.13) is satisfied with equality instead of inequality.

Combining this expression with the network stability upper bound, we get:

$$\bar{\lambda}_{\mathsf{GHOST}} \leq \min\left\{\beta \bar{f}_{\mathsf{GHOST}}(\beta), 1 - e^{\beta - 1}\right\}. \tag{3.15}$$

The throughput is plotted in Figure 3.6. As in Bitcoin, there are two regimes, communication-limited for β small, and security-limited for β large. Interestingly, the throughput of GHOST goes to zero as β approaches 0.5, just like Bitcoin. So although GHOST was invented to improve the throughput-security tradeoff of Bitcoin, the mining rate f still needs to vanish as β gets close to 0.5. The reason is that although GHOST is indeed secure against Nakamoto private attacks for any mining rate f [168], it is not secure against balancing attacks for f above a threshold as a function of β. When β is close to 0.5, this threshold goes to zero.

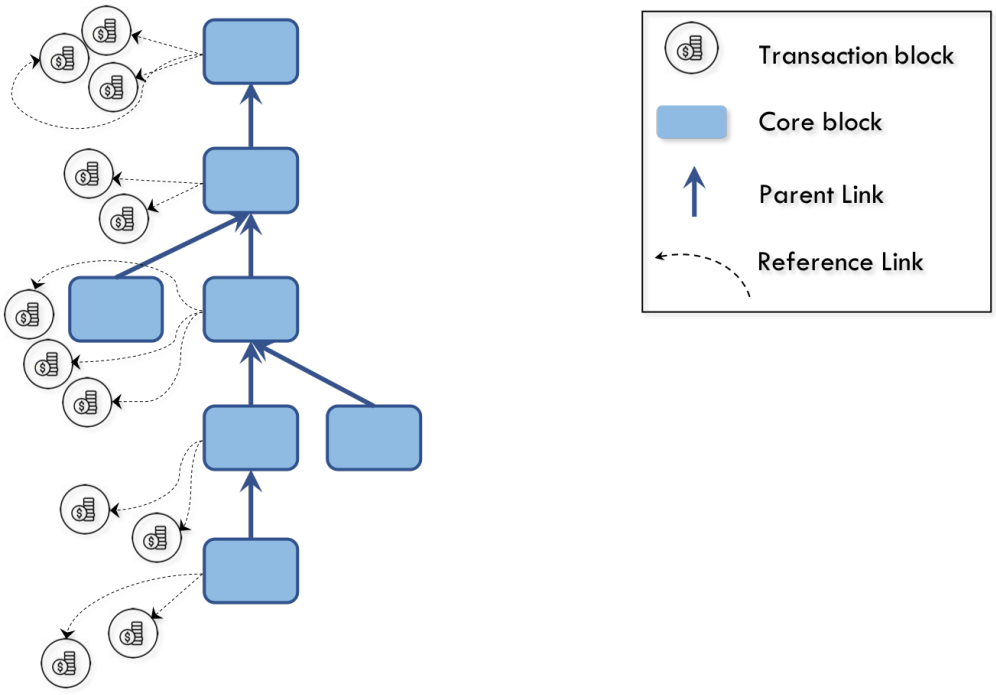


Figure 3.7: Prism 1.0. Decoupling the transaction blocks from the core blocks in the Bitcoin blockchain.

3.4.2 Prism 1.0: throughput-optimal protocol

We propose a protocol, Prism 1.0, that achieves optimal throughput efficiency $\bar{\lambda} = 1 - \beta$, which does not vanish as β approaches 0.5. We will build on Prism 1.0 in Section 3.5 to obtain our full protocol Prism.

Forking is the root cause of Bitcoin's and GHOST's inability to achieve optimal throughput. In designing Prism 1.0, our key insight is that we can create a secure blockchain by running Bitcoin at low mining rate with little forking, but incorporate additional transaction blocks, created via sortition, through reference links from the Bitcoin tree (Figure 3.7). This allows us to decouple the throughput from the mining rate f, and can increase the former without increasing the latter. In the context of the overall deconstruction approach (Figure 3.4), this decoupling is achieved by decoupling the transaction blocks from the core blockchain. Let us call the blocks in the core Bitcoin blockchain *core blocks*. Later, when we discuss latency, we will further split the functionalities of the core blocks into proposer and voter blocks to build a more complex consensus protocol, but for now we will just run Bitcoin as the basic consensus mechanism.

We now describe the structure of Prism 1.0.

1. There are two hash-threshold parameters α_c and α_t, such that $\alpha_c \leq \alpha_t$. A node mines blocks using a nonce. If the hash is less than the stringent threshold α_c, the block is a core block. If the hash is less than the relaxed threshold α_t but greater than α_c, the block is transaction block. This is a sortition of blocks into two types of blocks, and the adversary does not know which type of block it is mining until after the block has been mined.

2. The core blocks are used to determine the structure of the main chain. Each core block will reference several transaction blocks, that are then assumed to be incorporated into the ledger.

3. A block consists of the following data items.
   (a) Public key for reward
   (b) Transactions
   (c) The hash pointer to the current core block on which it is mining.
   (d) Hash pointers (references) to transaction blocks that the miner knows of and that have not been referenced in the current main chain.
   (e) Nonce, which is mined by miners.

If the block is a transaction block, then the hash-pointers to the current core block as well as the hash-pointers to transaction blocks are not used. If the block is a core block, then the list of transactions is not utilized. We note that the structure of the block shown in Figure 3.7 allows us to only package the information necessary for each type of block. The ordered list of transaction blocks is produced by ordering the transaction blocks in the order in which they are referenced by the core blocks in the main chain. For example, if the core blocks are $c_1, .., c_k$, and $R(c_i)$ denotes the list of referenced transaction blocks by c_i, then the ordered list is given by $R(c_1), .., R(c_k)$. From the ordered list of blocks, we produce an ordered list of transactions. From this ordered list of transactions, the ledger is produced by sanitizing this list to ensure that no transaction appears twice, and for every double spend, only the first spend is kept in the sanitized ledger.

Now that the key components of the protocol have been mentioned, we now explain how the protocol is run by various nodes.

- Each new transaction is broadcast to all the nodes in the network.
- Each node maintains a queue of outstanding transactions. The input to the queue is observed transactions. A transaction is cleared from the queue if the node knows of a transaction block containing the transaction.
- Each node maintains a blocktree of observed core blocks and transaction blocks.
- A node attempts to mine its new block(s) on top of the current longest chain of core blocks in its local blocktree.
  1. The node includes in its block all transactions from its transaction queue that are valid with respect to the ledger formed from the current longest chain of the core blocktree.
  2. The node gives reference links to all transaction blocks not currently referenced by its core main chain.
- A node that hears of a block determines its validity by checking the hash. Unlike in Bitcoin, there is no transaction validity check for a block, since the ledger is sanitized later.

In the context of the round-by-round synchronous model, the $H[r]$ honest blocks mined in the rth round are now split into $H^c[r] \sim \mathsf{Poiss}((1-\beta)f_c\Delta)$ honest core blocks and $H^t[r] \sim \mathsf{Poiss}((1-\beta)f_t\Delta)$ honest transaction blocks, where $f_c + f_t = f$. Similarly, the $Z[r]$ adversarial blocks mined in the rth round are split into $Z^c[r] \sim \mathsf{Poiss}(\beta f_c\Delta)$ adversarial core blocks and $Z^t[r] \sim \mathsf{Poiss}(\beta f_t\Delta)$ adversarial transaction blocks. The parameters f_c and f_t can be specified by choosing the appropriate value of the hash threshold α_c.

3.4.3 Analysis

We now analyze the proposed protocol in our network model. It is clear that the security of the protocol is the same as the security of the Bitcoin core blockchain. By setting f_c to be appropriately small (depending on β), we know that we can keep the core blockchain secure. More specifically, [79] gives one such sufficient condition, obtained by requiring that the rate of arrival of uniquely successful rounds exceeds the rate of work of the adversary:

$$f_c\Delta < \frac{1}{1-\beta}\ln\frac{1-\beta}{\beta} \tag{3.16}$$

Under this condition, [79] showed that the longest chain satisfies the common-prefix property as well as has positive chain quality. Similar to the argument in Conflux [114], the honest blocks in the longest chain can provide a total ordering of *all* the blocks, not just the core blocks. Hence, the throughput is given by the overall mining rate $f = f_c + f_t$. By choosing f_t such that we are at the capacity limit, i.e. $f = C/B$, we can get a total throughput of $(1-\beta)C/B$ blocks per second, or $(1-\beta)C$ transactions per second, assuming a worst case where only honest blocks carry transactions.

This seems to give us the optimal throughput efficiency $\bar{\lambda} = 1-\beta$. However, there is a catch: blocks that are mined in the same round may contain the same transactions, since transactions are broadcast to the entire network. In the worst case, we have to assume that blocks mined at the same round contain an identical set of transactions. In this case, mining more than one block per round does not add to the (useful) throughput. Hence, the throughput, in terms of number of non-redundant blocks, is simply:

$$\mathbb{P}(H[r] > 0) = 1 - e^{-(1-\beta)f\Delta} \quad \text{blocks per second.}$$

Comparing to (3.7), we see that this is exactly the longest chain growth rate of Bitcoin. Since Prism 1.0 can operate at $f = C/B$, we are achieving exactly the communication-limited throughput of Bitcoin (c.f. (3.12)), i.e.

$$\bar{\lambda} = 1 - e^{\beta-1}, \qquad \beta \in [0, 0.5).$$

The difference with the throughput-security tradeoff of Bitcoin is that Prism 1.0 is operating at the communication-limited regime for β all the way up to 0.5; there is no security-limited regime anymore. This is because we have decoupled transaction blocks from the core blockchain and the throughput is not security limited. In particular, the throughput does not go to zero as β goes to 0.5. But we are still not achieving the optimal throughput efficiency of $\bar{\lambda}^* = 1-\beta$.

3.4.4 Transaction scheduling

To achieve optimal throughput, one can minimize the transaction redundancy in the blocks by scheduling different transactions to different blocks. Concretely, one can split the transactions randomly into q queues, and each honest block is created from transactions drawn from one randomly chosen queue. Thinking of each transaction queue as a color, we now have transaction blocks of q different colors.

We will only have honest blocks with redundant transactions if two or more blocks of the same color are mined in the same round. The number of honest blocks of the same color mined at the same round is distributed as Poisson with mean $(1-\beta)f\Delta/q$, and so the throughput of non-redundant blocks of a given color is

$$1 - e^{-(1-\beta)f\Delta/q} \text{ blocks per round.}$$

The total throughput of non-redundant honest blocks of all colors is

$$q\left[1-e^{-(1-\beta)f\Delta/q}\right] \text{ blocks per round.} \tag{3.17}$$

For large q, this approaches

$$(1-\beta)f\Delta \text{ blocks per round,}$$

which equals $(1-\beta)C$ transactions per second when we set $f = C/B$. Thus, we achieve the optimal throughput efficiency

$$\bar{\lambda}^* = 1-\beta.$$

This performance is shown in the upper plot in Figure 3.6.

Interestingly, this maximum throughput of Prism 1.0 can be achieved whatever the choice of the block size B. In contrast, the block size B has to be set large compared to the bandwidth-delay product CD to optimize the throughput in both Bitcoin and GHOST. This extra degree of freedom in Prism 1.0 has significant implications on the tradeoff between throughput and transaction latency, which we turn to next.

3.4.5 Throughput-Latency tradeoff

So far we have focused on achieving the maximum throughput of Prism 1.0, without regard to latency. But transaction latency is another important performance metric. The overall latency experienced by a transaction in Prism 1.0 is the sum of two components:

1. **Processing latency** τ_p: the latency from the time the transaction enters the transaction queue to the first time a block containing that transaction is mined.
2. **Confirmation latency** τ: the latency from the time the transaction is first mined to the time it is confirmed.

We will discuss in great depth the confirmation latency in Section 3.5, but for now let us focus on the processing latency τ_p. It turns out that there is a tradeoff between the throughput λ and the processing latency τ_p.

We can calculate τ_p by considering the dynamics of an individual transaction queue. Let us make the simplifying assumption that transactions enter this queue at a deterministic rate. For a given total throughput λ and q, the number of transaction queues, the arrival rate into this queue is λ/q transactions per second. For stability, these transactions must also be cleared at a rate of λ/q. Thus it takes time qB/λ seconds to clear a block of B transactions from the queue and enters the blockchain. Hence,

$$\tau_p = \frac{qB}{\lambda} \quad \text{seconds.} \tag{3.18}$$

On the other hand, from (3.17), we see that the throughput λ, at the capacity limit, is given by

$$\lambda = q\left[1-e^{-(1-\beta)C\Delta/(Bq)}\right]\frac{B}{\Delta} \quad \text{transactions per second} \tag{3.19}$$

We see that increasing with the number of transaction queues q increases the throughput but also increases the processing latency, as the effective arrival rate decreases. Hence tuning q can effect a tradeoff between throughput and latency. To see the tradeoff explicitly, we can eliminate q from (3.18) and (3.17) and obtain:

$$\bar{\lambda} = \frac{1-\beta}{\bar{\tau}_p \log\left(\frac{1}{1-\frac{1}{\bar{\tau}_p}}\right)} \qquad 1 < \bar{\tau}_p < \infty, \tag{3.20}$$

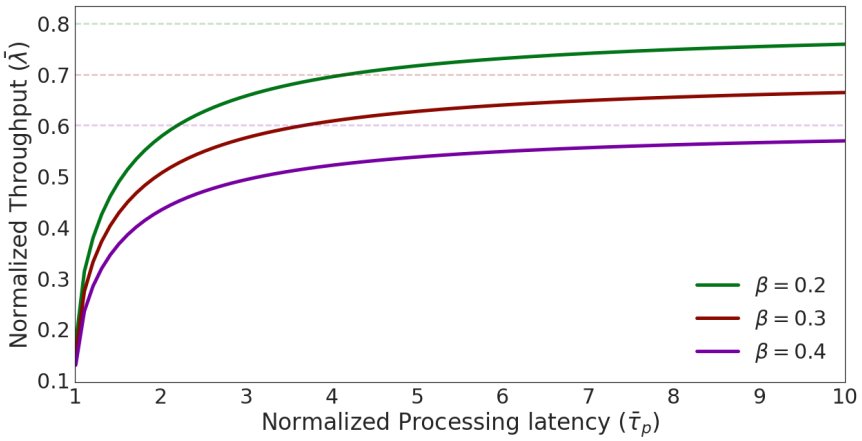


Figure 3.8: Tradeoff between $\bar{\lambda}$ and $\bar{\tau}_p$ for different values of β. Throughput is normalized as a fraction of the network capacity, and the processing latency is normalized as a multiple of the speed-of-light propagation delay.

where $\bar{\tau}_p := \frac{\tau_p}{\Delta}$.

We see that as $\bar{\tau}_p$ goes to infinity, the throughput efficiency $\bar{\lambda}$ approaches $1 - \beta$, the maximum throughput derived in previous section. This maximum throughput does not depend on the choice of the block size B, and this fact is consistent with our previous observation. However, for a given latency τ_p, the throughput achieved depends on the network delay Δ, which does depend on the block size B. By choosing the block size B small such that $B \ll CD$, Δ achieves the minimum value of the propagation delay D, optimizing the tradeoff. Under this choice of the block size B, (3.20) becomes a tradeoff between $\bar{\lambda}$, the throughput as a fraction of network capacity, and $\bar{\tau}_p$, the processing latency as a multiple of the propagation delay (Figure 3.8). Thus Prism 1.0 is achieving throughput and processing latency *simultaneously* near their physical limits. Note that Bitcoin and GHOST are not only sub-optimal in their maximum throughput, their throughput-latency tradeoff is also much worse. In particular, to achieve a non-zero throughput efficiency, the block size of these protocols is much larger than the bandwidth-delay product CD, and as a consequence, the processing latency of these protocols needs to be much larger than the propagation delay.

The remaining question is whether the *confirmation* latency can also be made close to the propagation delay. This is not the case in Prism 1.0 since its confirmation latency is the same as that of Bitcoin. This latency scales with $\log 1/\varepsilon$, where ε is the confirmation error probability; this security parameter $\log 1/\varepsilon$ can be many multiples of the network delay. The question is whether we can improve upon Prism 1.0 to make the confirmation latency of the same order as the processing latency. This will be addressed in Section 3.5.

3.4.6 Discussions

We discuss the relationships of our protocol with several existing protocols.

1. Conflux[114] separates *links* into two types: main-chain links and reference links, but all the *blocks* go into the same blocktree. As a result, the Conflux's security is limited by GHOST, and because Conflux is not done in conjunction with transaction scheduling, its throughput-β tradeoff is exactly the same as that of GHOST shown in Figure 3.6. In contrast, Prism 1.0 not only separates links into two types but also separates *blocks* into two types: core blocks, which

go into the core blockchain, and transaction blocks, which are referenced by the core blocks. This separation allows the protocol to have high security and high throughput simultaneously.

2. Prism 1.0 can be viewed as similar to BitcoinNG [63] but avoiding the risk of bribery attacks since the core block does not control which transactions to put into the ledger. Moreover, the core blocks incorporate transaction blocks from various nodes, thus increasing decentralization and fairness, unlike BitcoinNG where the leaders are entitled to propose blocks till a new leader comes up.

3. Fruitchains [137] was designed as a mechanism to increase reward fairness and Prism 1.0 is designed for a totally different purpose of maximizing throughput, but the structure of Prism 1.0 has similarity to Fruitchains. The transaction blocks are roughly analogous to fruits, though there are a few differences. The fruits hang-off an earlier block in Fruitchains for short-term reward equitability, but we do not need that for throughput optimality. The 2-for-1 mining protocol [79, 137] used in Fruitchains is somewhat different from our protocol. But more importantly, as we saw, transaction scheduling is crucial for achieving optimal throughput but is not present in Fruitchains.

4. Our two-threshold protocol is also similar to the ones used in mining pools [111]. Indeed, in mining-pools, partial Proof-of-Work using a higher hash threshold is used by participants to prove that they have been working (since they may be unable to generate full proof-of-work messages at regular intervals).

5. Our protocol is reminiscent of an approach called subchains [149] or weak blocks [20, 172]. Both methods employ blocks with lower hash threshold ("weak blocks") along with regular blocks. However, unlike our protocol, these weak blocks have to form a chain structure. Thus, if the PoW rate of weak blocks is increased significantly, it will lead to high forking on the weak blocks, thus leading to lower throughput.

6. We note that a version of transaction scheduling appears in Inclusive [112] for incentive compatibility. In order to maximize the reward gained, selfish users select a random subset of transactions to include in the ledger. In our protocol, we show this is required to maximize transaction throughput, even with altruistic users.

3.5 Near physical limits: latency and throughput

Prism 1.0 scales throughput to the network capacity limit by decoupling transaction blocks from the core blockchain, so that we can run Bitcoin on the core blockchain for high security and simultaneously maximize throughput by having many transaction blocks. However, the confirmation latency of Prism 1.0 is the same as Bitcoin, which is poor. In this section, our goal is to upgrade Prism 1.0 to design Prism, which has fast latency (on list ledger decoding and on honest transactions) as well as high throughput. The key idea is to further decouple the core blocks into proposer and voter blocks.

We start by describing the latency of Bitcoin, our baseline, in Section 3.5.1. In Section 3.5.2, we specify the Prism protocol. There are two parts to the specification: 1) the backbone (in the spirit of [79]), which specifies how the proposer blocks and voter blocks are organized, 2) how the transactions are linked from the proposer blocks. In Section 3.5.3, we provide a formal model for Prism based on a refinement of the model in Section 3.3. In Section 3.5.4, we prove several key properties of the Prism backbone, analogous to the common-prefix and chain-quality properties of Bitcoin proved in [79], and use it to show that it can achieve total ordering of all transactions and

has optimal throughput. Finally in 3.5.5, we show that Prism can achieve ledger list confirmation and honest transaction confirmation with fast latency.

3.5.1 Bitcoin latency

Bitcoin runs the longest chain protocol where each node mines blocks on the longest chain. These blocks have two roles: proposing to become a leader and voting on its ancestor blocks for them to be elected leaders. In this protocol, a current main chain block remains in the future main-chain with probability $1-\varepsilon$ if on the order of $\log 1/\varepsilon$ successive blocks are mined over it. It can be shown that at a mining rate of f, it takes on average [25]:

$$\mathbb{E}[\tau] = \frac{O(1)}{(1-2\beta)^2 f} \log \frac{1}{\varepsilon} \quad \text{seconds}$$

to provide $1-\varepsilon$ reliability to confirm blocks and the transactions in it. Since the expected latency τ is inversely proportional to the mining rate f, one might believe that increasing the mining rate will reduce latency. However, in the previous sections we have seen that naively increasing the mining rate will also increase forking, which reduces security in terms of β. To be more precise, Equation (3.8) limits the mining rate per round $\bar{f} := f\Delta$ to satisfy:

$$1 - e^{-(1-\beta)\bar{f}} > \beta\bar{f}.$$

For β close to 0.5, this leads to the following upper bound on $\bar{f}$:

$$\bar{f} < \frac{1-2\beta}{(1-\beta)^2}.$$

Therefore, this imposes a lower bound on the the expected latency of

$$\mathbb{E}[\tau] > \frac{O(1)\Delta(1-\beta)^2}{(1-2\beta)^3} \log \frac{1}{\varepsilon} \quad \text{seconds.} \tag{3.21}$$

Recall that physical limits impose two lower bounds on the latency: (1) the propagation delay D, and (2) $1/C \log 1/\varepsilon$. The above lower bound on Bitcoin latency is far from these physical limits, for two reasons. First, the network delay $\Delta = B/C + D$ depends on the block size B as well as the propagation delay. From the analysis of the throughput of Bitcoin, we know from (3.10) that to have decent throughput, the block size B should be chosen to be significantly larger than the bandwidth-delay product CD. But this implies that the network delay Δ is significantly larger than the propagation delay. Second, the Bitcoin latency lower bound's dependency on the security parameter is much larger than $1/C \log \frac{1}{\varepsilon}$. This is because the mining rate f of Bitcoin is limited by security and hence the voting rate is much less than what is allowed by the network capacity.

By decoupling transaction blocks from the blockchain, we learnt from our analysis of Prism 1.0 that we can choose the block size B small to keep the network delay near the speed-of-light propagation delay while achieving optimal throughput. Prism inherits this property of Prism 1.0, which overcomes the first reason why Bitcoin's latency is far from the physical limit. The focus of the remaining section is the design and analysis of a voting architecture to overcome the second issue, i.e. to increase the voting rate to the physical capacity limit.

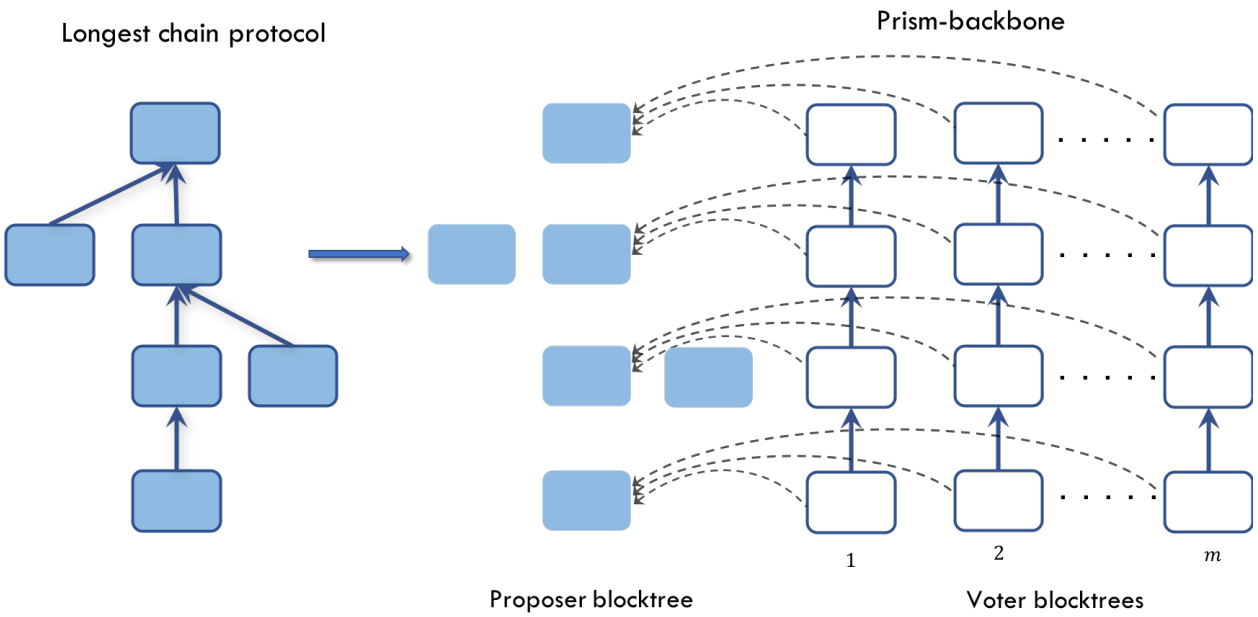


Figure 3.9: Prism : Separating proposer and voter roles.

3.5.2 Prism

3.5.2.1 Prism: backbone

We begin by describing Prism's backbone, or blockchain architecture; this architecture specifies how blocks relate to each other, and which blocks find a place in the final ledger. We describe how individual blocks are packed with transactions in Section 3.5.2.2. Each block in Bitcoin acts as both a proposer and a voter, and this couples their proposing and voting functionalities. As a result, the security requirements of the proposer role upper bounds the mining rate, which in turn upper bounds the voting rate. In the spirit of deconstructing the blockchain, we decouple these roles as illustrated in Figure 3.9. The backbone of Prism has two types of blocks: proposer and voter blocks. The role of the proposer block is to propose an extension to the transaction ledger. The voter blocks elect a leader block by voting among the proposer blocks at the same level. The sequence of leader blocks on each level determine the ledger. The voter blocks are mined on many independent blocktrees, each mined independently at a low mining rate. The voter blocktrees follow the longest chain protocol to provide security to the leader election procedure which in turn provides security to the transaction ledger. We now state the Prism backbone protocol from a node's local view:

- *Proposer blocks:* Proposer blocks are mined on a proposer blocktree as shown in Figure 3.9, using the longest-chain rule. The *level* of a proposer block is defined as the length of its path from the genesis block. Each proposer block includes a reference link to an existing proposer block to certify its level.

- *Voter blocks:* Voter blocks are mined independently on m separate voter trees, as shown in Figure 3.9. Each of these blocktrees has its own genesis block and nodes mine on the longest chain. Each voter block votes one or more proposer blocks using reference links.

- *Vote Interpretation:* Each voter blocktree votes only on one proposer block at each level in the proposer blocktree. The vote of the voter blocktree is decided by the vote cast by the earliest

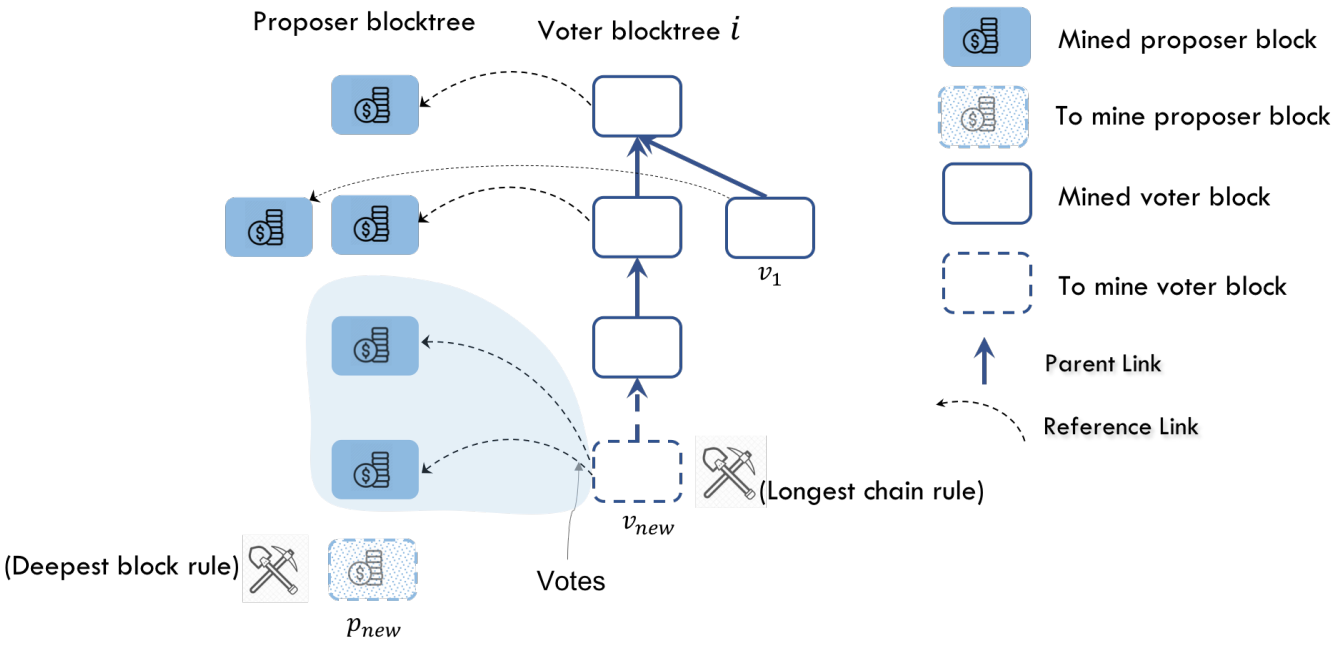


Figure 3.10: Prism: Honest users mine a proposer block p_{new} at a level one deeper than the current deepest level—in this example, p_{new} has level 5. The voter block v_{new} is mined on the longest chain. It votes (via reference links) to all proposer block on level $\{3, 4\}$ because its ancestors have votes only till level 2. Since v_1 is not part of the main chain, its vote will not be taken into account for leader block election.

voter block along its main chain. Thus the proposer blocks on each level has m votes in total. A voter block voting on multiple proposer blocks at the same level is invalid.

- *Voting rule:* The ancestor blocks of a voter block are all the blocks on its path to the genesis block. A voter block has to vote on a proposer block on all the levels which have not been voted by its ancestors voter blocks.

- *Leader blocks:* The proposer block that receives the most votes on each level is the) leader block for that level. The sequence of leader blocks across the levels is called the *leader sequence*.

- *Sortition:* A block is mined *before* knowing whether it will become a proposer block or a voter block. In case it becomes a voter block, the miner will not know a priori which voter tree it will be part of. This is enforced by using a sortition scheme, similar to the sortition described earlier in Prism 1.0 between core and transaction blocks, except now the hash range is divided into $m + 1$ instead of 2 intervals. This division is adjusted to ensure that the proposer tree has proposer rate f_p and each of the m voter trees have block mining rate f_v, with a total mining rate $f = f_p + mf_v$. By the security property of the hash function, a miner cannot know which range it will land in. This ensures that the adversarial power is uniformly distributed across the different voter trees and hence we assume the adversarial hash power is β in each of the voter trees as well as the proposer chain.

- *Choice of parameters:* Our protocol can operate with general settings of the parameters, but for good performance we set some specific numbers here. We set the block size $B = 1$ transaction, which as we discussed earlier is a good choice both for latency and for throughput. Under

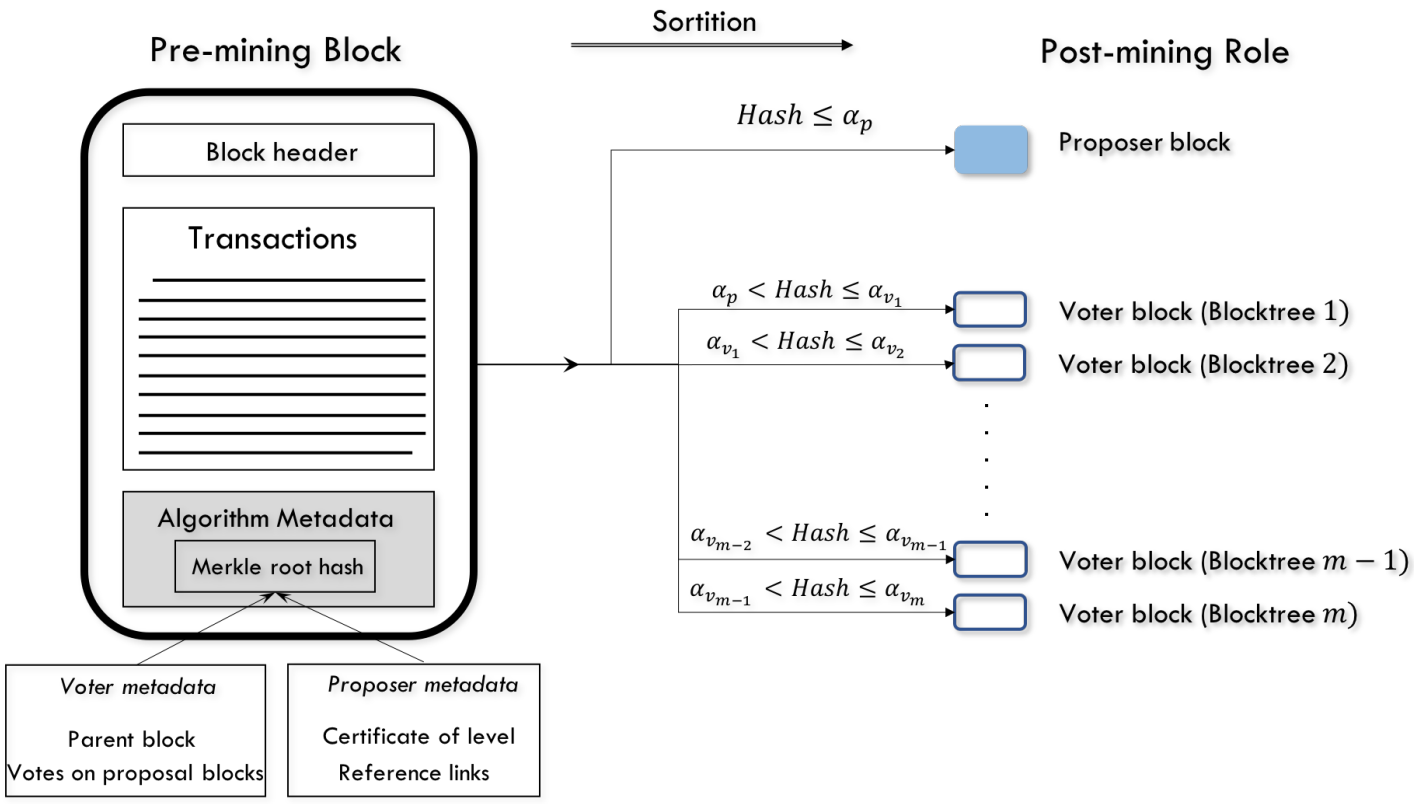


Figure 3.11: Summary of the block structure and the sortition procedure.

the assumption that $CD \gg 1$, the network delay $\Delta = D$, the smallest possible. To minimize latency, we want to maximize the vote generation rate, i.e. we set $f = C$, the capacity limit. The mining rate $\bar{f}_v := f_v D$ on each voting tree is chosen such that each voting tree is secure under the longest chain rule and according to (3.16) it should satisfy

$$\bar{f}_v < \frac{1}{1-\beta} \log \frac{1-\beta}{\beta}. \tag{3.22}$$

We also set the proposer and voter mining rates to be the same, i.e. $f_p = f_v$. This is not necessary but simplifies the notation in the sequel. This implies that

$$\begin{aligned} m &= \frac{CD}{\bar{f}_v} - 1 \\ &\geq \frac{(1-\beta)}{\log(\frac{1-\beta}{\beta})} \cdot CD - 1 \end{aligned} \tag{3.23}$$

i.e. the number of voting trees is at least proportional to the bandwidth-delay product CD. This number is expected to be very large, which is a key advantage of the protocol. The only degree of freedom left is the choice of $\bar{f}_v$. We will return to this issue in Section 3.5.5 when we discussed the fast confirmation latency of Prism.

3.5.2.2 Prism: transaction structure

Having presented the Prism backbone protocol, we now proceed to describe how transactions are embedded into this backbone structure. We also give more details on the content of the blocks. In

Prism, the structure of the block has to be fixed prior to determining whether the block will be a proposer-block or a voter-block; therefore both blocks will have the same fundamental structure.

Block contents: Any block needs to contain the following data items:

1. *Hash of Voter / Proposal Metadata* The block includes the hash of voter metadata as well as the hash of proposal metadata. Once it is known which type of block it becomes, then that particular metadata is attached to the block.

2. *Transactions*: Each block contains transactions that are not in the current ledger, and furthermore are not included in any of the referred blocks. The honest nodes utilize transaction scheduling given in Section 3.4.4 to choose a random subset of transactions.

3. *Nonce*: The nonce is a string discovered during PoW mining that seals the block; a valid nonce ensures that the hash value of the block (concatenated with the nonce) is less than a predetermined threshold. Our sortition mechanism uses the value of the hash to decide what type of block it becomes. In particular, we produce a sortition as follows:

   - Hash $< \alpha_p \Rightarrow$ Block is a proposer.
   - $(i-1)\alpha_v + \alpha_p <$ Hash $< i\alpha_v + \alpha_p \Rightarrow$ Block belongs to voter blocktree i .
   - The proposer PoW rate f_p will be proportional to α_p, and PoW rate on any voter blocktree f_v is proportional to α_v.

Voter Block Metadata: The voter block meta-data needs to contain two items: votes on the proposal blocks as well as where the parent block on the voter blocktree where it needs to be attached.

1. *Votes*: The votes are of the form (ℓ, p_ℓ) for $\ell \in \{\ell_{\min}, \ell_{\max}\}$ where p_ℓ is a hash of a proposer-block on level ℓ. The honest strategy is to vote on the block on level ℓ that it heard about the earliest. Also, for honest nodes $\ell_{\max}$ is the highest level that the node knows of, and $\ell_{\min}$ is the smallest level for which some blocktree has not yet voted.

2. *Parent link*: A voter block specifies one parent in each voter blocktree, $b_i, i = 1, 2, ..., m$. Honest nodes specify b_i as the leaf node in the longest chain of blocktree i. For efficiency, instead of storing all the m potential parents in the block, these potential parents are specified in a Merkle tree and only the root of the Merkle tree is specified in the block. If a block ultimately ends up in voter blocktree i, then it provides a proof of membership of b_i in the Merkle tree and is attached to voter block b_i.

Proposal Block Metadata: A proposal block needs to contain two metadata items, described as follows.

1. *Certificate of level*: A block that wants to be proposed for level ℓ contains a hash of a block in level $\ell - 1$.

2. *Reference links*: A proposal block p contains a list of reference links $\mathcal{R}(p)$ to other blocks. The honest strategy is to include a reference link to each proposal and voter block that is a leaf in the DAG. Here, the directed acyclic graph (DAG) is defined on the set of nodes equal to all the proposer and voter blocks. The edges include reference links from the proposer blocks to the voter blocks as well as the links from each voter block to its parent.

3.5.2.3 Generating the ledger

Given a sequence of proposer-blocks, $p_1, ..., p_\ell$, the ledger is defined as follows (our ledger construction procedure is similar to the one in Conflux [114]). Each proposer-block p_i defines an epoch; that epoch includes all the blocks referenced from that proposer block p_i, as well as all other blocks reachable from p_i but not included in the previous epochs. In each epoch the list of blocks is sorted topologically (according to the DAG), and ties are broken deterministically based on the content of the block. The ledger comprises the list of blocks ordered by epoch. Since the transactions in the reference blocks may have been mined independently, there may be redundant transactions or double-spends in the ledger of transactions. Any end-user can create a sanitized version of this ledger by keeping only the first time a given transaction output is spent. We note that this approach decouples transaction validation from mining, unlike in Bitcoin, where nodes only include valid transactions with respect to the current ledger.

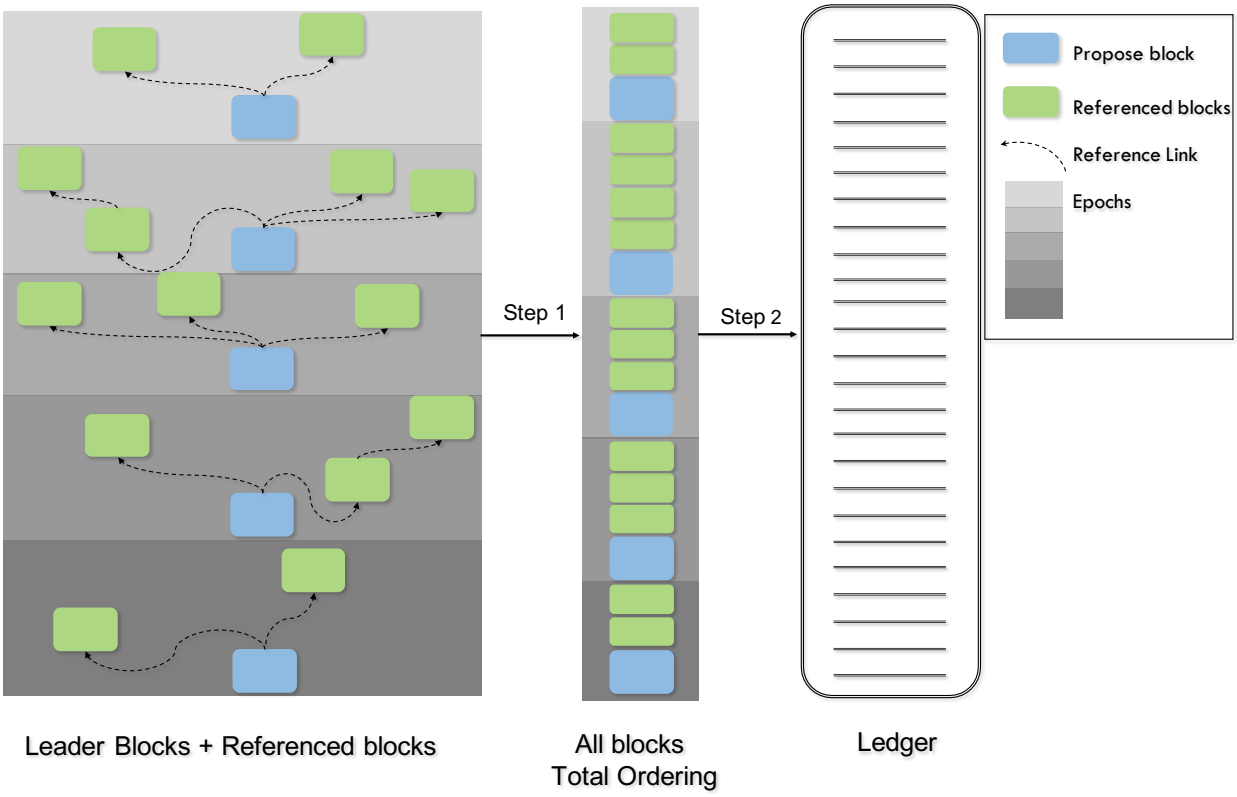


Figure 3.12: Prism: Generating the ledger. The proposer blocks for a given proposer block sequence are highlighted in blue, and the referenced blocks are shown in green. Each shade of grey corresponds to an epoch. In Step 1, all the blocks are incorporated, and in Step 2, they are expanded out to give a list of transactions.

3.5.3 Prism: model

We provide a formal model of Prism based on a refinement of the round-by-round synchronous model in Section 3.3.

Let $H_i[r]$ and $Z_i[r]$ be the number of voter blocks mined by the honest nodes and by the adversarial nodes in round r on the i-th voting tree respectively, where $i = 1, 2, .., m$. Note that

by the sortition process, $H_i[r], Z_i[r]$ are Poisson random variables with means $(1-\beta)f_v\Delta$ and $\beta f_v\Delta$ respectively, and are independent, and independent across trees and across rounds. Similarly, $H^p[r], Z^p[r]$ are the numbers of proposer blocks mined by the honest nodes and by the adversarial nodes in round r respectively, they are also Poisson, with means $(1-\beta)f_p\Delta$ and $\beta f_p\Delta$ respectively. They are independent, and independent of all the other random variables. We will define $X_i[r]$ ($X^p[r]$), which is 1 if $H_i[r] \geq 1$ ($H^p[r] \geq 1$) and zero otherwise. and define $Y_i[r]$ ($Y^p[r]$), which is 1 if $H_i[r] = 1$ ($H^p[r] = 1$) and zero otherwise. We denote $H_i[r_1, r_2] := \sum_{r=r_1+1}^{r_2} H_i[r]$, similarly for Z_i, X_i and Y_i. The interval $[r_1, r_2]$ denotes rounds $\{r_1+1, r_1+2, \cdots, r_2-1, r_2\}$.

The adversary decides to release blocks (either kept in private or just mined) in each tree (either the proposer tree or one of the voter trees) after observing all the blocks mined by the honest nodes in all trees in that round. It can also decide which proposal block each honest voter votes on (but it cannot remove the vote or violate protocol, e.g., by changing the proposal block level of the vote.) The adversary is powerful as it can observe what is happening in *all* the trees to make a decision on its action in any individual tree. In particular, this adversarial power means that the evolution of the trees are *correlated* even though the mining processes on the different trees are independent because of the sortition procedure. This fact makes the analysis of Prism more subtle, as we need to prove some kind of law of large numbers across the voter trees, but can no longer assume independence.

As in our basic model (which follows [79]), all the nodes have a common view of the (public) trees at the end of each round.

3.5.4 Total transaction ordering at optimal throughput

In this subsection, we show that Prism can achieve total transaction ordering for any $\beta < 0.5$. Following the framework of [79], we do so by first establishing two backbone properties: common-prefix and quality of a certain *leader sequence* of proposer blocks, analogous to the longest chain under Bitcoin.

The blockchain runs for $r_{\max}$ rounds, which we assume to be polynomial in m i.e, $r_{\max} = \text{poly}(m)$. Let $\mathcal{P}(r)$ denote the set of proposer blocks mined by round r. Let $\mathcal{P}_\ell(r) \subseteq \mathcal{P}(r)$ denote the set of proposer blocks mined on level ℓ by round r. Let the first proposer block on level ℓ be mined in round R_ℓ. Let $V_p(r)$ denote the number of votes on proposer block $p \in \mathcal{P}(r)$ at round r. Recall that only votes from the main chains of the voting trees are counted. The *leader block* on level ℓ at round r, denoted by $p^*_\ell(r)$, is the proposer block with maximum number of votes in the set $\mathcal{P}_\ell(r)$ i.e,

$$p^*_\ell(r) := \underset{p \in \mathcal{P}_\ell(r)}{\text{argmax}}\, V_p(r),$$

where tie-breaking is done in a hash-dependent way.

A *proposer sequence* up to level ℓ at round r is given by $[p_1, p_2, \cdots, p_\ell]$, where $p_j \in \mathcal{P}_j(r)$. The *leader sequence* up to level ℓ at round r, denoted by $\texttt{LedSeq}_\ell(r)$, is a proposer sequence with $p_j = p^*_j(r)$, in other words

$$\texttt{LedSeq}_\ell(r) := [p^*_1(r), p^*_2(r), \cdots, p^*_\ell(r)]. \tag{3.24}$$

The leader sequence at the end of round $r_{\max}$, the end of the horizon, is the *final leader sequence*, $\texttt{LedSeq}_\ell(r_{\max})$.

The leader block $p^*_\ell(r)$ for a fixed level ℓ can change with round r due to the adversary displacing some of the votes from their voter chains. However as r increases, changing $p^*_\ell(r)$ is harder as the votes are embedded deeper in their respective voter chains. The theorem below characterizes this phenomenon.

Theorem 1 (Leader sequence common-prefix property). *Suppose $\beta < 0.5$. For a fixed level ℓ, we have*

$$\texttt{LedSeq}_\ell(r) = \texttt{LedSeq}_\ell(r_{\max}) \quad \forall r \geq R_\ell + r(\varepsilon) \tag{3.25}$$

with probability $1-\varepsilon$, where $r(\varepsilon) = \frac{1024}{\bar{f}_v(1-2\beta)^3} \log \frac{8mr_{\max}}{\varepsilon}$, and R_ℓ is the round in which the first proposer block on level ℓ was mined.

Proof. See Appendix in [25]. □

Theorem 1 is similar in spirit to Theorem 15 of [79], which establishes the common-prefix property of the longest chain under the Bitcoin backbone protocol. Hence, the leader sequence in Prism plays the same role as the longest chain in Bitcoin. Note however that the leader sequence, unlike the longest chain, is not determined by parent-link relationships between the leader blocks. Rather, each leader block is individually determined by the (many) votes from the voter chains.

The common-prefix property of Bitcoin's longest chain guarantees consistency of the ledger produced by the protocol. Similarly, the common-prefix property of the leader sequence guarantees consistency of the ledger produced by Prism. Ledger liveness of Bitcoin, on the other hand, is guaranteed by the chain-quality property. The proposer block mining policy (Section 3.5.2.1) is to mine each proposer block at the highest level available, with a reference link to a parent block that certifies the new block's level. If we define a tree with proposer blocks and these reference links as the edges, then the users are in fact mining over the longest proposer chain. Therefore, intuitively, the chain-quality guarantees of Theorem 16 in [79] should hold for the leader sequence, resulting in the liveness of the Prism ledger. This result is formalized below and proved in [25].

Theorem 2 (Liveness). *Assume $\beta < 0.5$. Once a transaction enters into a mined block, w.p $1-\varepsilon$ it will eventually be pointed to by a permanent leader sequence block after a finite expected latency.*

Together, Theorem 1 and Theorem 2 guarantee that Prism achieves a consistent and live total ordering of all transactions, but requiring a confirmation latency of order $\log \frac{m}{\varepsilon}$ for a confirmation error probability of ε. Just like the longest chain in the core tree of Prism 1.0, the leader sequence blocks of Prism orders all the transactions in the transaction blocks they refer to. In conjunction with transaction scheduling, Prism, just like Prism 1.0, achieves a worst-case optimal throughput of $(1-\beta)C$ transactions per second.

While being able to achieve a total ordering of transactions at optimal throughput is an important property of a consensus protocol, this goal was already accomplished in the simpler Prism 1.0, using the longest chain protocol on the core tree. The use of a more sophisticated voting structure in Prism is to meet a more ambitious goal: fast confirmation latency near physical limit. We turn to this goal in the next subsection.

3.5.5 Fast confirmation of ledger list and honest transactions

3.5.5.1 An example

Let us start with an example to get a feel for why we can confirm with latency much shorter than Bitcoin latencies.

Suppose $CD = 5000$, $D = 0.2$ seconds and $\bar{f}_v = 0.1$ (corresponding to a tolerable $\beta = 0.49$), so we have $m \approx 5000/0.1 = 50,000$ votes at each level and votes are mined at rate $1-e^{-\bar{f}_v} = 1-e^{-0.1} \approx 0.1$ votes per round per voter chain. Two proposer blocks are mined from genesis at round 1 and appear in public at level 1. At the next round, on average, $0.1 \cdot 50,000 = 5000$ votes are generated to vote on these two proposer blocks. At the round after that, only the voter chains that have not voted in

the last round can generate new votes, and on the average $0.1 \cdot (50000 - 5000) = 4500$ votes will be generated. The total number of chains that have not voted after r rounds is:

$$m(1 - 0.1)^r,$$

decreasing exponentially with r. After 20 rounds, or 4 seconds, about 6000 chains have not voted. That means at least one of the two proposer blocks has at least $(50,000 - 6000)/2 = 22,000$ votes.

At this point:

1. If the adversary later presents a proposer block that it has mined in private at this level, then it can gather at most 6000 votes and therefore not sufficient to displace both these two public blocks and become a leader block. Thus, no private attack is possible, and we are ensured that anytime in the future one of the two proposer blocks already in public will be a leader block.

2. If one of the public proposer blocks has significantly more votes than the other block, by much more than 6000, then we can already confirm that the current leader block will remain the leader forever, because there are not enough new votes to change the ordering.

Interestingly, when these events occur, an observer observing the public blockchain knows that it occurs. Moreover, we know that the first event will definitely occur after r rounds, where r is the smallest number of rounds such that

$$m(1 - 0.1)^r < \frac{m - m(1 - 0.1)^r}{2},$$

i.e. $r = 12$ rounds.

The above analysis gives some evidence that fast confirmation is possible, but the analysis is simplistic, due to three reasons:

1. $1 - e^{-\bar{f}_v}$ is the growth rate of each voter chain if every node follows the protocol. However, some fraction of the voter blocks on the chains may belong to adversarial nodes who decide not to vote on any proposer block; in fact, this fraction may be greater than β due to selfish mining [64, 156, 133]. Thus, the number of outstanding votes calculated above may be on under-estimation. However, we do know that the longest chain's quality is non-zero (Theorem 16 of [79]). Hence, the qualitative behavior of the voting dynamics remain the same but the voting rate has to be reduced to account for adversarial behavior.

2. The above analysis assumes that votes that have already been cast cannot be reversed. That is not true because the adversary can grow private chains to reverse some of the votes. However, because the adversary power is limited, the fraction of such votes that can be reversed is also limited. Moreover, as we wait longer, the fraction of votes that can be reversed in the future also gets smaller because the votes get embedded deeper in their respective chains. This needs to be accounted for, but again the qualitative picture from the simplistic analysis remains unchanged: after waiting for a finite number of rounds, one can be sure that the eternal leader block will be one of a list of current public proposer blocks.

3. The simplistic analysis assumes the total number of votes that are mined at each round is *deterministic*, at the mean value. In reality, the actual number of votes mined at each round is random, fluctuating around the mean value. However, due to a law of large numbers effect, which we will formally show, the fluctuations will be very small, since there are large number of voting chains. This justifies a deterministic view of the dynamics of the voting process.

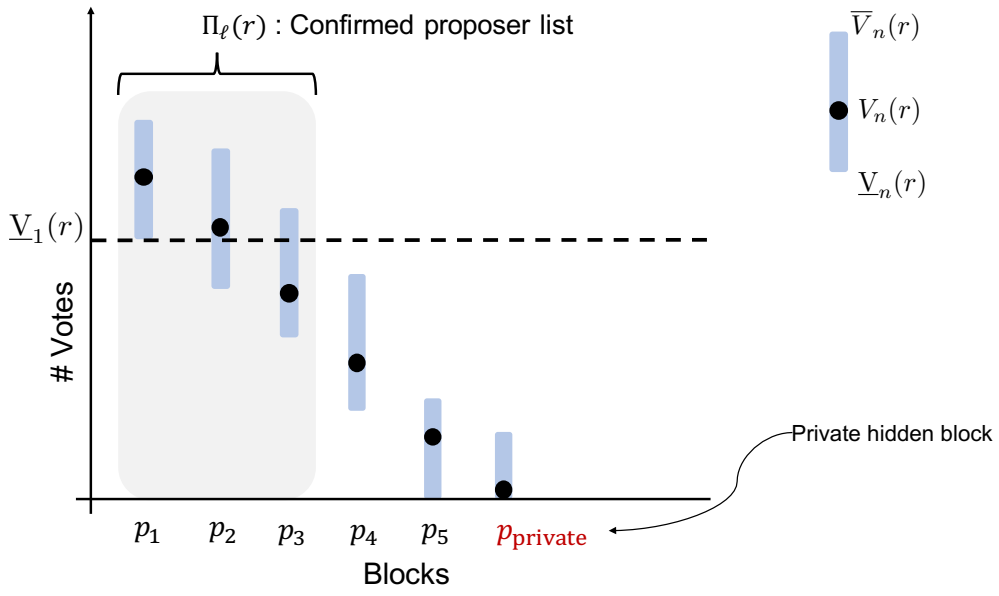


Figure 3.13: In the above example, public proposer block p_1 has the largest lower confidence bound, which is larger than the upper confidence bound of the private block. So list confirmation is possible and the list confirmed is $\Pi_\ell(r) = \{p_1, p_2, p_3\}$.

3.5.5.2 Fast list confirmation

We convert the intuition from the above example to a formal rule for fast confirming a *list* of proposer blocks, which then allows the confirmation of a list of proposer sequences. The idea is to have *confidence intervals* around the number of votes cast on each proposer block. Figure 3.13 gives an example where there are 5 proposal blocks in public at a given level, and we are currently at round r. The confidence interval $[\underline{V}_n(r), \overline{V}_n(r)]$ for the votes on proposer block p_n bounds the maximum number of votes the block can lose or gain from votes not yet cast and from the adversary reversing the votes already cast. In the running there is also potentially a private hidden block, with an upper bound on the maximum number of votes it can accumulate in the future. We can fast confirm a list of proposal blocks whenever the upper confidence bound of the private block is below the lower confidence bound of the public proposal block with the largest lower confidence bound.

More formally: Let $\mathcal{P}_\ell(r) = \{p_1, p_2 ...\}$ be the set of proposer blocks at level ℓ at round r. Let $U(r)$ be the number of voter blocktrees which have not voted for any proposer block in $\mathcal{P}_\ell(r)$. Let $V_n^d(r)$ be the number of votes at depth d or greater for proposer block p_n at round r. Let $V_{-n}^d(r)$ be the the number of votes at depth d or greater for a proposer blocks in the subset $\mathcal{P}_\ell(r) - \{p_n\}$. Define:

$$\delta_d := \max\left(\frac{1}{4\bar{f}_v d}, \frac{1-2\beta}{8\log m}\right)$$

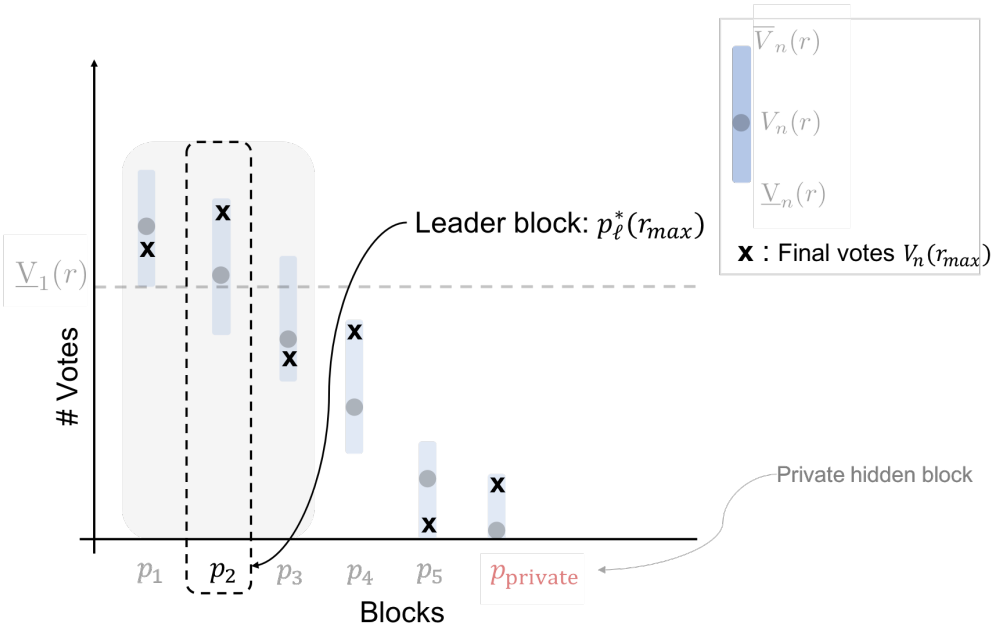


Figure 3.14: A possible scenario by the final round.

and

$$\underline{V}_n(r) := \max_{d\geq 0}\left(V_n^d(r) - 2\delta_d m\right)_+,$$
$$\overline{V}_n(r) := V_n(r) + \left(V_{-n}(r) - \max_{d\geq 0}\left(V_{-n}^d(r) - 2\delta_d m\right)_+\right) + U(r),$$
$$\underline{V}_{\text{private}}(r) := 0,$$
$$\overline{V}_{\text{private}}(r) := m - \sum_{p_n\in\mathcal{P}_\ell(r)} \underline{V}_n(r).$$

Proposer list confirmation policy: If

$$\max_n \underline{V}_n(r) > \overline{V}_{\text{private}}(r),$$

then we confirm the list of proposer blocks $\Pi_\ell(r)$, where

$$\Pi_\ell(r) := \{p_n : \overline{V}_n(r) > \max_i \underline{V}_i(r)\}.$$

The following theorem shows that one can confirm proposer lists up to level ℓ with an expected latency independent of ε; moreover the final leader sequence is contained in the product of the confirmed lists.

Theorem 3 (List common-prefix property). *Suppose $\beta < 0.5$. Suppose the first proposer block at level ℓ appears at round R_ℓ. Then w.p. $1 - r_{\max}^2 e^{-\frac{(1-2\beta)m}{16\log m}}$, we can confirm proposer lists $\Pi_1(r), \Pi_2(r), \ldots, \Pi_\ell(r)$ for all rounds $r \geq R_\ell + R_\ell^{conf}$, where*

$$\mathbb{E}[R_\ell^{conf}] \leq \frac{2808}{(1-2\beta)^3\bar{f}_v}\log\frac{50}{(1-2\beta)} + \frac{256}{(1-2\beta)^6\bar{f}_v m^2}. \tag{3.26}$$

Moreover, w.p. $1 - r_{\max}^2 e^{-\frac{(1-2\beta)m}{16\log m}}$,

$$p^*_{\ell'}(r_{\max}) \in \Pi_{\ell'}(r) \quad \forall \ell' \le \ell \text{ and } r \ge R_\ell + R^{conf}_\ell.$$

Proof. See Appendix of [25]. □

Let us express the latency bound (3.26) in terms of physical parameters. If we set the voting rate $\bar{f}_v$ equal to the largest possible given the security constraint (3.22):

$$\bar{f}_v = \frac{1}{1-\beta}\log\frac{1-\beta}{\beta},$$

then according to (3.23), we have

$$m = \frac{(1-\beta)}{\log(\frac{1-\beta}{\beta})} \cdot CD - 1.$$

With this choice of parameters, and in the regime where the bandwidth-delay product CD is large so that the second term in (3.26) can be neglected, the expected latency for list confirmation is bounded by

$$c_1(\beta)D \quad \text{seconds},$$

i.e. proportional to the propagation delay. Here,

$$c_1(\beta) := \frac{2808(1-\beta)}{(1-2\beta)^3\log\frac{1-\beta}{\beta}}\log\frac{50}{(1-2\beta)}$$

and is positive for $\beta < 0.5$. The confirmation error probability is exponentially small in CD. This is the constant part of the latency versus security parameter tradeoff of Prism in Figure 3.1.

Since CD is very large in typical networks, a confirmation error probability exponentially small in CD is already very small. To achieve an even smaller error probability ε we can reduce the voting rate $\bar{f}_v$ smaller below the security constraint (3.22) and increase the number of voting chains. More specifically, we set

$$\bar{f}_v = \frac{CD}{\log\frac{1}{\varepsilon}}, \tag{3.27}$$

resulting in

$$m = \log\frac{1}{\varepsilon} - 1 \approx \log\frac{1}{\varepsilon}$$

yielding the desired security parameter. Indeed, the above equation for $\bar{f}_v$ is valid only if the security condition for $\bar{f}_v$ is satisfied, i.e., when $\varepsilon < e^{-\frac{CD(1-\beta)}{\log\frac{1-\beta}{\beta}}}$.

Again neglecting the second term in (3.26), the corresponding latency bound is

$$\frac{c_2(\beta)}{C}\log\frac{1}{\varepsilon} \quad \text{seconds},$$

where

$$c_2(\beta) := \frac{2808}{(1-2\beta)^3}\log\frac{50}{(1-2\beta)}.$$

This is the linearly increasing part of the tradeoff curve for Prism in Figure 3.1, with slope inversely proportional to the network capacity C.

Some comments:

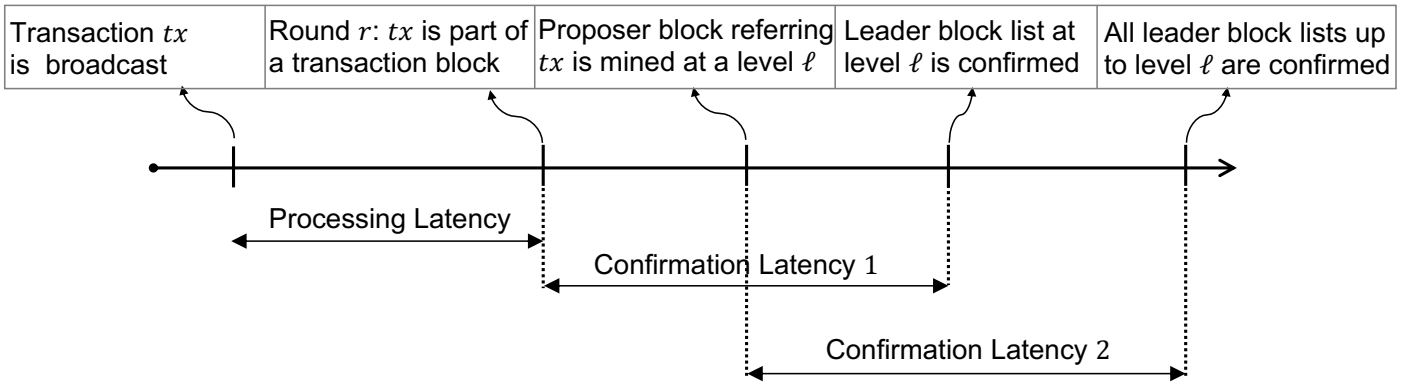


Figure 3.15: Components of the latency: a) Processing latency is addressed in Section 3.4.5, b) Confirmation latency 1 is analyzed in Theorem 4, and c) Confirmation latency 2 is analyzed in Theorem 3.

- We have shown that latency and confirmation reliability can be traded off by choosing different values of $\bar{f}_v$ and m. But these are protocol parameters. We believe that one can achieve a similar tradeoff by changing the *confirmation rule* while fixing these protocol parameters. This would allow the recipient of a transaction to choose the level of security guarantee that they require and wait accordingly. A detailed analysis of this adaptive confirmation rule is left for future work.
- While the latency bounds exhibit the correct qualitative behavior, the constants involved are rather large. This is due to two reasons. First, our proofs are optimized for clarity rather than yielding the best constants. In particular, we structure the proofs to mirror as close as possible the backbone protocol framework of [79]. Second, in our analysis, we give full power to the adversary in choosing which proposer blocks the honest voter blocks vote on. Thus the bounds need to account for the worst case, where the number of votes on the proposer blocks are very close. With a less crude model, one can improve the bounds considerably. We expect the actual latency to be much smaller than our bounds, but this conjecture is best validated by experiments rather than more theory.

3.5.5.3 Fast confirmation of honest transactions

In the previous subsection we have shown that one can fast confirm a list of proposer block sequences which is guaranteed to contain the prefix of the final totally ordered leader sequence. As discussed in Section 3.5.2.3, each of these proposer block sequence creates an ordered ledger of transactions using the reference links to the transaction blocks. In each of these ledgers, double-spends are removed to sanitize the ledger. If a transaction appears in *all* of the sanitized ledgers in the list, then the transaction is guaranteed to be in the final total ordered sanitized ledger, and the transaction can be fast confirmed. (See Figure 3.5.) All honest transactions without double-spends eventually have this *list-liveness* property; When only a single honest proposer block appears in a level and becomes the leader, it will add any honest transactions that have not already appeared in at least one of the ledgers in the list. Due to the positive chain-quality of the leader sequence (Theorem 2, this event of "uniquely honest" level eventually occurs. The latency of confirming honest transactions is therefore bounded by the sum of the latency of list confirmation in Theorem 3 plus the latency of waiting for this event to occur(Figure 3.15. The latter is given by the following theorem.

Theorem 4 (List-liveness)**.** *Assume $\beta < 0.5$. If a honest transaction without double spends is mined in a transaction block in round r, then w.p. $1 - r_{\max}^2 e^{-\frac{m}{16 \log m}}$ it will appear in all of the ledgers corresponding to proposer block sequences after an expected latency no more than*

$$\frac{2592}{(1-2\beta)^3 \bar{f}_v} \log \frac{50}{(1-2\beta)} \quad \textit{rounds.}$$

The proof is available in Appendix of [25].

Figure 3.15 shows the various components of the overall latency we analyzed. We can see that the confirmation latency from the time an honest transaction enters a blocks to the time it is confirmed is bounded by the sum of the latencies in Theorem 3 and 4. Repeating the analysis in the previous subsection, we see that this latency is bounded by:

$$\max\{a_1(\beta)D, \frac{a_2(\beta)}{C} \log \frac{1}{\varepsilon}\} \quad \text{seconds,}$$

where

$$a_1(\beta) := \frac{5400(1-\beta)}{(1-2\beta)^3 \log \frac{1-\beta}{\beta}} \log \frac{50}{(1-2\beta)} \tag{3.28}$$

$$a_2(\beta) := \frac{5400}{(1-2\beta)^3} \log \frac{50}{(1-2\beta)}. \tag{3.29}$$

3.6 Discussions

3.6.1 Prism: incentives

Our discussion on Prism has mostly focussed on honest users and adversarial behavior. Here we briefly discuss rational behavior, and the accompanying reward structure that incentivizes rational users to participate in the system without deviating from the proposed protocol. There are straightforward approaches to add a reward structure to Prism. Each block, whether a voter block or a proposal block, that finds its place in the ledger is assigned a block reward. To allocate transaction fees, we follow the method proposed in Fruitchains [137]. The transaction fees are distributed among the past Q blocks, where Q is a design parameter. In Prism, all blocks eventually find a place in the ledger, and thus the proportion of blocks contributed by a miner to the ledger is proportional to the hash rate of the miner. For large values of Q, our design ensures that incentives are fairly distributed and there is no gain in pursuing selfish-mining type attacks [156].

3.6.2 Prism: smart contracts

Most of our discussion on Prism has focused on transactions. However, we point out here that Prism is not restricted to processing transactions and can be extended to process complex smart contracts. Smart contracts are pieces of code which are executed based on the current state of the ledger. Importantly, they can depend on the *history of the ledger*, including on the timing of various events recorded on the ledger. While many of the basic blockchain protocols such as longest-chain consensus or GHOST protocol can accommodate smart contracts, newer schemes such as Spectre and Avalanche are specific to transactions and do not confirm smart contracts. We note that Prism is naturally able to confirm the output and final-state of every smart contract at the ε-dependent latency since we get total order. We also note that this is the behavior desired in hybrid algorithms like Phantom+ Spectre .

We note that Prism has an additional attractive property for smart contracts - the ability to confirm several smart contracts at a short latency (proportional to propagation delay). Since Prism is able to confirm a list of ledgers within a short latency, this can be exploited to confirm some smart contracts. If a smart contract will execute to the same final state and output in all the ledgers in this list, then this output and final state can be confirmed for the smart contract even before confirming a unique ledger. We recall that Prism guarantees short confirmation time for honest transactions. Analogous to the notion of honest transactions, we can define a notion of *uncontested smart contracts*, where there is no alternate view of how the events happened in any of the blocks. Such uncontested smart contracts can then be shown to be confirmed within a short ε-independent latency proportional to the propagation delay - thus enhancing the scope and utility of Prism beyond payment systems.

3.6.3 Prism: Proof-of-Stake

In this paper we have described Prism in the proof-of-work (PoW) setting that scales the throughput by three orders of magnitude over Bitcoin . Despite this significant increase, PoW is nevertheless energy inefficient (Bitcoin consumes as much energy as medium sized countries [55]) and a leading alternative is the so-called proof-of-stake (PoS) paradigm. PoS restricts involvement in the consensus protocol to nodes that deposit a requisite amount of stake, or currency, into the system. This stake is used as a security deposit in case the nodes misbehave – for instance, by trying to unduly influence the outcome of consensus. PoS is appealing for several reasons, including the fact that it can be much more energy-efficient than PoW and also because it can be more incentive-compatible.

There are two key issues associated with designing a PoS version of Prism. First, a cryptographically secure source of randomness, that is distributed and verifiable, is needed to replace the source of randomness currently used in Prism – this includes the various mining steps, transaction scheduling and sortition operations. Second, PoS does not have the conservation of work that is implicit in PoW and this allows adversaries to "mine" at no cost in parallel and only report the outcomes that can be successfully verified – this exposes new security vulnerabilities (popularly known as the "grinding" [3] and "nothing at stake" attacks [116, 101]) and a PoS design of Prism will have to contend with this attack. Both these obstacles can be successfully surmounted and will be the topic of a forthcoming paper [23].

3.7 Acknowledgement

We thank Mohammad Alizadeh and Andrew Miller for their comments on an early draft. The chapter includes material in [25].

Chapter 4

Barracuda: Consensus-Aware P2P Networking

Giulia Fanti, CMU
Jiantao Jiao, UC Berkeley
Sewoong Oh, UW Seattle
Pramod Viswanath, UIUC

The previous chapter presented Prism, a consensus protocol that achieves latency and throughput approaching the physical limits of a *given network*. The network's properties could be determined by many factors, ranging from hardware to relaying algorithms, but the properties were ultimately treated as static. This view of peer-to-peer (P2P) networks is very common in the blockchain community: the network is treated as a static interface over which consensus protocols can be built. To some extent, this is a good approximation; it is difficult and expensive to change the network's hardware, for example. However, other aspects of the network are more flexible, like the network topology and the relaying algorithms. In this chapter, we discuss a method for improving the throughput of a given consensus algorithms by co-designing the corresponding networking protocols. To understand these innovations, we start by summarizing some of the modeling decisions from Chapter 3.

In designing Prism, we made several implicit assumptions about the behavior of the underlying P2P network. In particular, we assumed a model of network block propagation that is based on a very common *push-based* design for P2P networks. In push-based relay networks, blocks are communicated to the rest of the network (roughly) as follows: each peer with a given block attempts to push the block to its connected peers. If a target recipient has not seen the block yet, it downloads the block, checks that the block is valid, and starts the relay process anew with its own peers. Our communication models in Chapter 3 are implicitly based on such a push-based relay network, whose delay characteristics were measured by Decker and Wattenhofer in [58]. This assumption manifests itself in two parameters from Chapter 3: the network delay D and the network capacity C. Since our fundamental upper bound on throughput is precisely $(1-\beta)C$ transactions/second for adversarial fraction β, increasing C can enable higher throughput.

In practice, there is no fundamental reason to use a push-based P2P network. Indeed, it is well-known in the distributed systems literature that push-based protocols are suboptimal for spreading information in a network [96]. In this chapter, we show how to boost the performance of a given consensus mechanism on a given network by simply changing the relay mechanism. We introduce a protocol called Barracuda, which combines the usual push-based relay mechanism with a lightweight

polling primitive, in which select nodes selectively poll their neighbors to learn of new blocks [67]. We show that for a broad class of consensus mechanisms, this simple modification leads to substantial improvements in throughput; the final effect is to artificially boost the speed of the network. Notice that while Chapter 3 proposed an entirely new consensus protocol, this chapter asks how to improve the performance of a *given* consensus protocol using techniques at a lower layer—the network layer. For example, Barracuda can be combined with Bitcoin's Nakamoto consensus just as it can be combined with PRISM. To explain the problem more precisely, we start with a brief description of blockchain functionality; to make the chapter self-contained, this description will contain some material that was discussed in Chapter 3.

4.0.1 Primer

The core problem in blockchain systems is determining (and agreeing on) the next block in the data structure. Many leading cryptocurrencies (e.g., Bitcoin, Ethereum, Cardano, EOS, Monero) handle this problem by electing a *proposer* who is responsible for producing a new block and sharing it with the network. This proposer election happens via a distributed, randomized protocol chosen by the system designers. For example, in Bitcoin, mining is the proposer election protocol.

In proof-of-stake (PoS) systems, a proposer is elected with probability proportional to their stake in the system. This election process typically happens at fixed time intervals (e.g., every 10 seconds). When a node is elected proposer, its job is to propose a new block, which contains a hash of the previous block's contents. Hence the proposer must choose where in the blockchain to append her new block. Most blockchains use a *longest chain* fork choice rule, under which the proposer always appends her new block to the end of the longest chain of blocks in the proposer's local view of the blocktree. If there is no network latency and no adversarial behavior, this rule ensures that the blockchain will always be a perfect chain. However, in a network with random delays, it is possible that the proposer may not have received all blocks when she is elected. As such, she might propose a block that causes the blockchain to *fork* (e.g. Figure 4.2). In longest-chain blockchains, this forking is eventually resolved with probability 1 because one fork eventually overtakes the other.

Forking occurs in almost all major blockchains, and it implies that blockchains are often not chains at all, but *blocktrees*. For many consensus protocols (particularly chain-based ones like Bitcoin's), forking reduces throughput, because blocks that are not on the main chain are discarded. It also has security implications; even protocols that achieve good block throughput in the high-forking regime have thus far been prone to security vulnerabilities (which has been resolved in a recent work [24], which also guarantees low latency). Nonetheless, forking is a significant obstacle to *practical* performance in existing blockchains. There are two common approaches to mitigate forking. One is to improve the network itself, e.g. by upgrading hardware and routing. This idea has been the basis for recent projects like the Falcon network [4] and Bloxroute [102]. The other is to design consensus algorithms that tolerate network latency by making use of forked branches. Examples include GHOST [168], SPECTRE [167], and Inclusive/Conflux [112, 114]. In this paper, we design a P2P protocol called Barracuda that effectively reduces forking for a wide class of *existing* consensus algorithms.

4.0.2 Contributions

Our contributions in this work are threefold.

1. We propose a probabilistic model for the evolution of a blockchain in proof-of-stake cryptocurrencies, where the main source of randomness comes from the network delay. This captures the network delays measured in real world P2P cryptocurrency networks [59]. This model has

two main differences with the model in Chapter 3. First, it is random, whereas the model in Chapter 3 was worst-case. Second, it models the delay of each block to each individual recipient, whereas the model from Chapter 3 models the worst-case delay to *all* recipients. Simulations under our new model explain the gap observed in real-world cryptocurrencies between the achievable block throughput and the best block throughput possible in an infinite-capacity network. Our model differs from that of prior theoretical papers, which typically assume a worst-case network model that allows significant simplification in the analysis [79, 168]. We analyze the effect of average network delay on system throughput and provide a lower bound on the block throughput.

2. To mitigate the effect of forking from network delays, we propose a new block proposal algorithm called *Barracuda*, under which nodes poll ℓ randomly-selected nodes for their local blocktree information before proposing a new block. We show that for small values of ℓ, Barracuda has approximately the same effect as if the *entire network* were a factor of ℓ faster. Notably, this benefit emerges without actually changing any inherent network properties, like bandwidth or hardware. The analysis also has connections to load balancing in balls-and-bins problems, which is of independent interest.

3. We provide guidelines on how to implement Barracuda in practice in order to provide robustness against several real-world factors, such as network model mismatch and adversarial behavior.

Outline. We begin by describing a stochastic model for blocktree evolution in Section 4.2; we analyze the block throughput of this model in Section 4.3. Next, we present Barracuda and analyze its block throughput in Section 4.4. We describe real-world implementation issues in Section 4.5, such as how to implement polling and analyzing adversarial robustness. Finally, we discuss how to integrate Barracuda with Prism in Section 4.6.

4.1 Related Work

This section will discuss related approaches for reducing forking. It overlaps substantially with the related work of Chapter 3 (Section 3.2), but is included here for completeness. There are three main approaches in the literature for reducing forking. The first is to reduce the diversity of proposers. The second is to embrace forking, and use the forks to enhance throughput. The third is to algorithmically resolve forking before moving to the next block.

(1) Reducing proposer diversity. Forking is caused by the delay associated with the most recently-proposed block reaching the next proposer(s). A natural circumvention is to make the proposer(s) of consecutive blocks one and the same. This approach is proposed by Bitcoin-NG [63]: proposers use the longest-chain fork choice rule, but within a given time epoch, only a single proposer can propose blocks. This allows the proposer to quickly produce blocks without worrying about network delay or forking. Although Bitcoin-NG has high throughput, it exhibits a few problems. First, whenever a single node is in charge of block proposal for an extended period of time, attackers may be able to learn that node's IP address and take it down. Second, it suffers from high confirmation delay. To confirm a transaction in a longest-chain protocol, we require a threshold number of independently-selected proposers to append blocks to the chain containing that transaction. Since Bitcoin-NG elects a new proposer only once every epoch, this takes time comparable to Nakamoto consensus. Despite these problems, the idea of having a fixed proposer is

also used in other protocols, such as Thunderella [140] and ByzCoin [104], which are also vulnerable to attacks on the proposer's IP address.

(2) Embracing forking. A different class of protocols has studied how to use forked blocktrees to contribute to throughput. Examples include GHOST [168], PHANTOM [166], SPECTRE [167], and Conflux [114]. GHOST describes a fork choice rule that tolerates honest forking by building on the heaviest subtree of the blocktree; it is described more carefully in Section 4.2.2. SPECTRE, PHANTOM, and Conflux instead use existing fork choice rules, but build a directed acyclic graph (DAG) over the produced blocks; this DAG is used to define an ordering over transactions. A formal understanding of these DAG-based protocols is still evolving; although the ideas are intuitively appealing, their security properties are not yet well-understood.

(3) Structured DAGs. A related approach is to allow *structured* forking. The Prism consensus mechanism explicitly co-designs a consensus protocol and fork choice rule to securely deal with concurrent blocks, thereby achieving optimal throughput and latency [24]. The key intuition is to run many concurrent blocktrees, where a single proposer tree is in charge of ordering transactions, and the remaining voter trees are in charge of confirming blocks in the proposer tree. Our approach differs from [24] in that Barracuda is designed to be integrated into *existing* consensus protocols, whereas Prism is a completely new consensus protocol. Indeed, since each of the blocktrees in Prism uses the longest-chain fork choice rule, Barracuda can be used to reduce forking in each individual Prism blocktree.

(4) Fork-free consensus. Several consensus protocols tackle forking by preventing it entirely. Examples include Algorand [45, 82], Ripple [11], and Stellar [120]. These systems conduct a full round of consensus for every block, e.g., using voting-based protocols. Disagreements about the next block are resolved immediately. Although voting-based consensus protocols consume additional time upfront for each block, the hope is that they improve overall efficiency by removing the need to resolve forks later; this hypothesis remains untested. A primary challenge in such protocols is that Byzantine-fault tolerant voting protocols can be communication-intensive, and require a known set of participants. Although there has been work addressing some of these challenges [104, 138], many industrial-grade blockchain systems running on BFT voting protocols require some degree of centralization for efficiency.

Our approach. Barracuda is complementary to these prior approaches in the sense that it provides a new networking protocol that consensus algorithms can benefit from. Our approach can be viewed as a partial execution of a polling-based consensus protocol. Polling has long been used in classical consensus protocols [50, 14, 75], as well as more recent work specific to blockchains [150]. Our approach differs from these algorithms in part because we do not use polling to reach complete consensus; rather we use it to reduce the number of inputs to a (separate) consensus protocol. Hence, Barracuda can be used as an add-on for many consensus protocols, especially those proposed for proof-of-stake (PoS) cryptocurrencies; we discuss these applications in Section 4.5.

4.2 Model

We propose a probabilistic model for blocktree evolution with two sources of randomness: randomness in the timing and the proposer of each new block, and the randomness in the delay in transmitting messages over the network. The whole system is parameterized by the number of nodes n, average

propagation delay Δ, proposer waiting time $\tilde{\Delta}$, and number of concurrent proposers k. Notice that this Δ is an *average* propagation delay, whereas the Δ from Chapter 3 represented a *worst-case* delay.

4.2.1 Modeling block generation

We model block generation as a discrete-time arrival process, where the t^{th} block is generated at time $\gamma(t)$. We previously discussed the election of a single proposer for each block; in practice, some systems elect multiple proposers at once to provide robustness if one proposer fails or is adversarial. Hence at time $\gamma(t)$, k nodes are chosen uniformly at random as *proposers*, each of which proposes a distinct block. The index $t \in \mathbb{Z}^+$ is a positive integer, which we also refer to as *time* when it is clear from the context whether we are referring to t or $\gamma(t)$. The randomness in choosing the proposers is independent across time and of other sources of randomness in the model. We denote the k blocks proposed at time t as $(t,1),(t,2),\ldots,(t,k)$. The block arrival process follows the distribution of a certain point process, which is independent of all other randomness in the model.

Two common block arrival process are Poisson and deterministic. Under a Poisson arrival process, $\gamma(t) - \gamma(t-1) \sim \mathsf{Exp}(\lambda)$ for some constant λ, and $\gamma(t) - \gamma(t-1)$ is independent of $\{\gamma(i)\}_{i=1}^{t-1}$. In proof-of-work (PoW) systems like Bitcoin, block arrivals are determined by independent attempts at solving a cryptographic puzzle, where each attempt has a fixed probability of success. With high probability, one proposer is elected each time a block arrival occurs (i.e., $k = 1$), and the arrival time can be modeled as a Poisson arrival process.

In many PoS protocols (e.g., Cardano [100], Qtum [9], and Particl [8]), time is split into quantized intervals. Some protocols give each user a fixed probability of being chosen to propose the next block in each time interval, leading to a geometrically-distributed block arrival time. If the probability of selecting *any* proposer in each time slot is smaller than one, the expected inter-block arrival time will be greater than one, as in Qtum and Particl. Other protocols explicitly designate one proposer per time slot (e.g., Cardano [39]). Assuming all nodes are active, such protocols can be modeled with a deterministic interval process, $\gamma(t) = t$, for all $t \in \mathbb{N}$. The deterministic arrival process may even be a reasonable approximation for certain parameter regimes of protocols like Qtum and Particl. If the probability of electing any proposer in a time step is close to one, there will be at least one block proposer in each time slot with high probability, which can be approximated by a deterministic arrival process. Regardless, our main results apply to arbitrary arrival processes $\gamma(t)$, including geometric and deterministic.

When a block (t,i) is generated by a proposer, the proposer attaches the new block to one of the existing blocks, which we refer to as the *parent block* of (t,i). The proposer chooses this parent block according to a pre-determined rule called a *fork-choice rule*; we discuss this further in Section 4.2.1. Upon creating a block, the proposer broadcasts a message containing the following information:

$$M_{t,i} = (\text{Block } (t,i), \text{pointer to the parent block of } (t,i))$$

to all the other nodes in the system. The broadcasting process is governed by our network model, which is described in Section 4.2.1.

In this work, we focus mainly on the PoS setting due to subtleties in the practical implementation of Barracuda (described in Section 4.4). In particular, PoW blockchains require candidate proposers to choose a block's contents—including the parent block—*before* generating the block. But in PoW, block generation itself takes an exponentially-distributed amount of time. Hence, if a proposer were to poll nodes before proposing, that polling information would already be (somewhat) stale by the time the block gets broadcast to the network. In contrast, PoS cryptocurrencies allow block creation to happen *after* a proposer is elected; hence polling results can be simultaneously incorporated into

a block and broadcast to the network. Because of this difference, PoS cryptocurrencies benefit more from Barracuda than PoW ones.

Global view of the blocktree. Notice that the collection of all messages forms a rooted tree, called the *blocktree*. Each node represents a block, and each directed edge represents a pointer to a parent block. The root is called the *genesis block*, and is visible to all nodes. All blocks generated at time $t = 1$ point to the genesis block as a parent. The blocktree grows with each new block, since the block's parent must be an existing block in the blocktree; since each block can specify only one parent, the data structure remains a tree. Formally, we define the global blocktree as follows.

Definition 4.2.1 (Global tree)**.** *We define the* global tree *at time t, denoted as G_t, to be a graph whose edges are described by the set $\{(Block\ (j,i), pointer\ to\ the\ parent\ block\ of\ (j,i)) : 1 \leq j \leq t, 1 \leq i \leq k\}$ with the vertices being the union of the genesis block and all the blocks indexed as $\{(j,i) : 1 \leq j \leq t, 1 \leq i \leq k\}$.*

If there is no network delay in communicating the messages, then all nodes will have the same view of the blocktree. However, due to network delays and the distributed nature of the system, a proposer might add a block before receiving all the previous blocks. Hence, the choice of the parent node depends on the local view of the blocktree at the proposer node.

Local view of the blocktree. Each node has its own local view of the blocktree, depending on which messages it has received. Upon receiving the message $M_{t,i}$, a node updates its local view as follows. If the local view contains the parent block referred in the message, then the block t is attached to it. If the local view does not contain the parent block, then the message is stored in an *orphan cache* until the parent block is received. Notice that G_t is random and each node's local view is a subgraph of G_t.

4.2.2 Network model and fork choice rule

We avoid modeling the topology of the underlying communication network by instead modeling the end-to-end delay of a message from any source to any destination node. In particular, we assume each transmission of a block over an edge experiences a delay distributed as an independent exponential random variable with mean Δ. This exponential delay captures the varying and dynamic network effects of real blockchain networks, as empirically measured in [59] on Bitcoin's P2P network. In particular, this exponential delay encompasses both network propagation delay and processing delays caused by nodes checking message validity prior to relaying it. These checks are often used to protect against denial-of-service attacks, for instance.

When a proposer is elected to generate a new block at time $\gamma(t)$, the proposer waits for time $\tilde{\Delta} \in [0, 1)$ to make a decision on where to append the new block in its local blocktree. The choice of parent block is governed by the local fork choice rule. Two of the most common fork choice rules are the Nakamoto protocol (longest chain) and the GHOST protocol.

- **Nakamoto protocol (longest chain).** When a node is elected as a proposer, the node attaches the block to the leaf of the *longest chain* in the *local* blocktree. When there is a tie (i.e., if there are multiple longest chains), the proposer chooses one arbitrarily. Longest chain is widely-used, including in Bitcoin, ZCash, and Monero.

- **GHOST protocol.** When a node is elected a proposer, the node attaches the block to the leaf of the *heaviest subtree* in the *local* blocktree. Concretely, the proposer starts from the genesis block and traverses the tree toward the leaves, until it reaches a leaf block. At each node,

the proposer chooses the offspring with the largest number of descendants (i.e., the heaviest subtree). Ties are broken arbitrarily. A variant of the GHOST rule is used in Ethereum.

Both the Nakamoto and GHOST protocols belong to the family of *local attachment protocols*, where the proposer makes the decision on where to attach the block solely based on the snapshot of its *local* tree at time $\gamma(t) + \tilde{\Delta}$, stripping away the information on the proposer of each block. In other words, we require that the protocol be invariant to the identity of the proposers of the newly generated block. We show in Section 4.4 that our analysis applies generally to all local attachment protocols. In practice, almost all blockchains use local attachment protocols; however, some recent blockchains (e.g., Ripple) construct each new block through a cooperative process between a group of nodes.

Notice that if Δ is much smaller than the block inter-arrival time and all nodes obey protocol, then the global blocktree G_t will be a chain with high probability. On the other hand, if Δ is much larger than the block inter-arrival time, then G_t will be a star (i.e. a depth-one rooted tree) with high probability. To maximize blockchain throughput, it is desirable to design protocols that maximize the expected length of the longest chain of G_t. Intuitively, a faster network infrastructure with a smaller Δ implies less forking.

4.3 Block Throughput Analysis

A key performance metric in blockchains is *transaction throughput*, or the number of transactions that can be processed per unit time. Transaction throughput is closely related to a property called *block throughput*, also known as the main chain growth rate. Given a blocktree G_t, the length of the main chain $L(G_t)$ is defined as the number of hops from the genesis block to the farthest leaf. More precisely,

$$L(G_t) \triangleq \max_{B \in \partial(G_t)} d(B_0, B),$$

where $\partial(G_t)$ denotes the set of leaf blocks in G_t, and $d(B_0, B)$ denotes the hop distance between two vertices B_0 and B in G_t. We define *block throughput* as $\lim_{t\to\infty} \mathbb{E}[L(G_t)]/t$. Block throughput describes how quickly blocks are added to the blockchain; if each block is full and contains only valid transactions, then block throughput is proportional to transaction throughput. In practice, this is not the case, since adversarial strategies like selfish mining [65] can be used to reduce the number of valid transactions per block. Regardless, block throughput is frequently used as a stepping stone for quantifying transaction throughput [168, 79, 24].

For this reason, a key objective of our work is to quantify block throughput, both with and without polling. We begin by studying block throughput without polling under the Nakamoto protocol fork-choice rule, as in Bitcoin. This has been previously studied in [168, 79, 24], under a simple network model where there is a fixed deterministic delay between any pair of nodes. This simple network model is justified by arguing that if all transmissions of messages are guaranteed to arrive within a fixed maximum delay d, then the worst case of block throughput happens when all transmissions have delay of exactly d. Such practice ignores all the network effects, for the sake of tractable analysis. In this section, we focus on capturing such network effect on the block throughput. We ask the fundamental question of how block throughput depends on the average network delay, under a more realistic network model where each communication is a realization of a random exponential variable with average delay Δ. In the following (Theorem 5), we provide a lower bound on the block throughput, under the more nuanced network model from Section 4.2, and Nakamoto protocol fork-choice rule. This result holds for a deterministic arrival process. A proof is provided in Appendix 4.7.1.

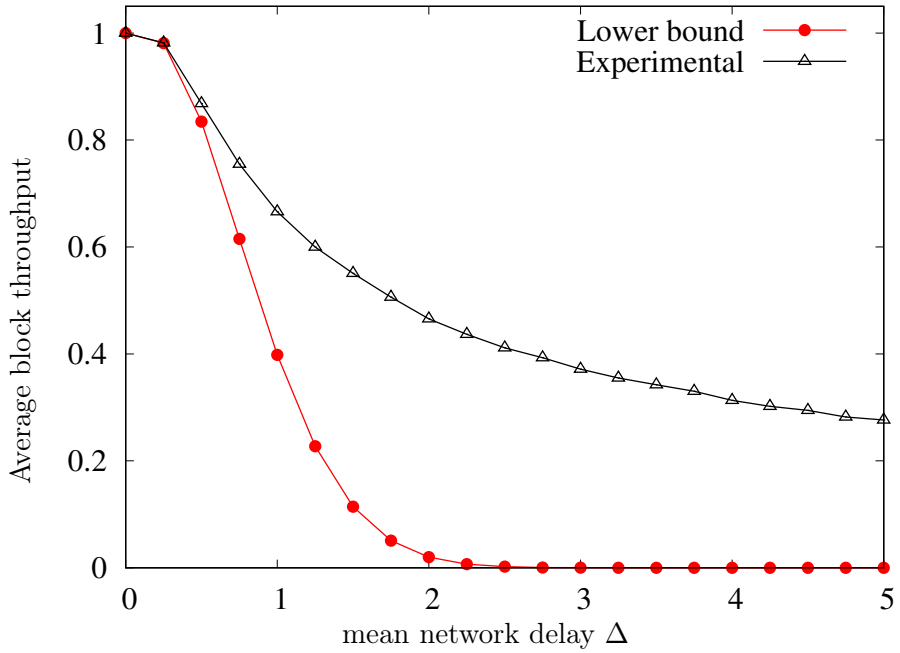


Figure 4.1: Block throughput vs. mean network delay for an inter-block time of 1 time unit.

Theorem 5. *Suppose there is a single proposer ($k = 1$) at each discrete time, $\gamma(t) = t \in \{1, 2, \ldots\}$, with no waiting time ($\tilde{\Delta} = 0$). For any number of nodes n, any time t, and any average delay Δ, under the Nakamoto protocol, we have that*

$$\frac{\mathbb{E}[L_{\mathsf{Chain}}(G_t)]}{t} \geq \exp\left(\frac{-C_\Delta}{(1-C_\Delta)^2}\right),$$

where $C_\Delta = e^{\frac{-1}{\Delta}}$.

Notice that trivially, $\mathbb{E}[L_{\mathsf{Chain}}(G_t)]/t \leq 1$, with equality when there is no network delay, $\Delta = 0$. Theorem 6 and our experiments in Figure 4.1 suggest that Theorem 5 is tight when $\Delta \ll 1$. Hence there is an (often substantial) gap between the realized block throughput and the desired upper bound. This gap is caused by network delays; since proposers may not have an up-to-date view of the blocktree due to network latency, they may append to blocks that are not necessarily at the end of the global main chain, thereby causing the blockchain to *fork*.

One goal is to obtain a blocktree with no forking at all, i.e., a perfect blockchain with $L_{\mathsf{Chain}}(G_t) = t$. Setting $\exp\left(\frac{-C_\Delta}{(1-C_\Delta)^2}\right) = 1 - \frac{1}{t}$, which implies that $\mathbb{E}[L_{\mathsf{Chain}}(G_t)] \geq t - 1$, we obtain that $\Delta = \Theta(\frac{1}{\log t})$. The following result shows that if $\Delta = O(\frac{1}{\log t})$, then $L_{\mathsf{Chain}}(G_t) = t$ with high probability.

Theorem 6. *Fix a confidence parameter $\delta \in (0, 1), k = 1, \gamma(t) = t$. For both the Nakamoto and GHOST protocols, if*

$$\frac{1}{\Delta} \geq \frac{\left(\ln t - \log\log\frac{1}{\delta}\right)}{1 - \tilde{\Delta}}, \tag{4.1}$$

then the probability of the chain $Gen - 1 - 2 - \ldots - t$ happens with probability at least $\delta - o(1)$ as $t \to \infty$ and $n \gg t^2$.

Conversely, when $n \gg (\Delta t \log t)^2$ *and*

$$\frac{1}{\Delta} \leq \frac{\left(\ln t - \log\log \frac{1}{\delta}\right)}{1-\tilde{\Delta}}, \tag{4.2}$$

then the probability of the chain $Gen - 1 - 2 - \ldots - t$ *happens with probability at most* $\delta + o(1)$ *as* $t \to \infty$*. Here* $\gg$ *ignores the dependence on the parameter* δ*, which is fixed throughout.*

This result shows the prevalence of forking. For example, if we conservatively use Bitcoin's parameters settings, taking $\Delta = 0.017$, $\tilde{\Delta} = 0$, and $\delta = 0.01$, equation (4.2) implies that for $t \gtrsim 5$ blocks, forking occurs with high probability. Hence forking is pervasive even in systems that choose system parameters specifically to avoid it.

A natural question is how to reduce forking, and thereby increase block throughput. To this end, we next introduce a blockchain evolution protocol called Barracuda that effectively reduces forking without changing the system parameter Δ, which is determined by network bandwidth.

4.4 ℓ-Barracuda

To reduce forking and increase block throughput, we propose ℓ-Barracuda, which works as follows: upon arrival of a block (t, i), the proposer of block (t, i) selects $\ell - 1$ nodes in the network uniformly at random, and inquires about their local tree.[1] The proposer aggregates the information from the $\ell - 1$ other nodes and makes a decision on where to attach block (t, i) based on the *local attachment protocol* it follows. One key observation is that there is no conflict between the local trees of each node, so the Barracuda strategy simply merges totally ℓ local trees into a single tree with union of all the edges in the local trees that are polled. Note that we poll $\ell - 1$ nodes, such that a total ℓ local trees are contributing, as the proposers own local tree also contributes to the union.

We assume that when Barracuda polling happens, the polling requests arrive at the polled nodes instantaneously, and it takes the proposer node time $\tilde{\Delta}$ to make the decision on where to attach the block. To simplify the analysis, we also assume that each node processes the additional polled information in real time, but does not store the polled information. In other words, the information a node obtains from polling at time t is forgotten at time $t' > t + \tilde{\Delta}$. This modeling choice is made to simplify the analysis; it results in a lower bound on the improvements due to polling since nodes are discarding information. In practice, network delay affects polling communication as well, and we investigate experimentally these effects in Section 4.5.1.

To investigate the effect of polling on the block chain, we define appropriate events on the probabilistic model of block arrival and block tree growth. We denote $X \sim \mathsf{Exp}(\lambda)$ an exponential random variable with probability density function $p_X(t) = \lambda e^{-\lambda t}\mathbb{1}(t \geq 0)$, and define set $[m] \triangleq \{1, 2, \ldots, m\}$ for any integer $m \geq 1$. For a message

$$M_{j,i} \;=\; (\text{Block } (j,i), \text{point to the parent block of } (j,i)),$$

denote its arrival time to node m as $R_{(j,i),m}$. If m is the proposer of block (j, i), then $R_{(j,i),m} = \gamma(j) + \tilde{\Delta}$. If m is not the proposer of block (j, i), then $R_{(j,i),m} = \gamma(j) + \tilde{\Delta} + B_{(j,i),m}$, where $B_{(j,i),m} \sim \mathsf{Exp}(1/\Delta)$. It follows from our assumptions that the random variables $B_{(j,i),m}$ are mutually independent for all $1 \leq j \leq t, 1 \leq i \leq k, 1 \leq m \leq n$. We also denote the proposer of block (j, i) as $m_{(j,i)}$. To denote polled nodes, we also write $m_{(j,i)}$ as $m^{(1)}_{(j,i)}$, and denote the other $\ell - 1$ nodes polled by node $m_{(j,i)}$ as $m^{(2)}_{(j,i)}, m^{(3)}_{(j,i)}, \ldots, m^{(\ell)}_{(j,i)}$.

[1] We use the name Barracuda to refer to the general principle, and ℓ-Barracuda to refer to an instantiation with polling parameter ℓ.

When block (j, i) is being proposed, we define the following random variables. Let random variable

$$e_{j,i,l,r} = \begin{cases} 1 & \text{if by the time } (j,i) \text{ was proposed, node} \\ & \quad m^{(l)}_{(j,i)} \text{ already received block } r \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Here $j \in [t], i \in [k], l \in [\ell], r \in \{(a, b) : a \in [j-1], b \in [k]\}$. For any $r = (a, b)$, we denote $r[1] = a, r[2] = b$.

Since we will aggregate the information from the total ℓ nodes whenever a proposer proposes, we also define $e_{j,i,r} = 1 - \prod_{l=1}^{\ell}(1 - e_{j,i,l,r})$ as the event that when (j, i) was proposed, at least one node $m^{(l)}_{(j,i)}$ has received block r. The crucial observation is that when the proposer tries to propose block (j, i), the complete information it utilizes for decision is the collection of random variables

$$\{e_{j,i,r} : r[1] \in [j-1], r[2] \in [k]\}. \tag{4.4}$$

The global tree at time $\gamma(t) + \tilde{\Delta}$, denoted as G_t, is a tree consisting of $kt + 1$ blocks including the Genesis block. We are interested in the distribution of the global tree G_t. To illustrate how to compute the probability of a certain tree structure, we demonstrate the computation through an example where $k = 1, t = 3$, and $\ell = 1$.

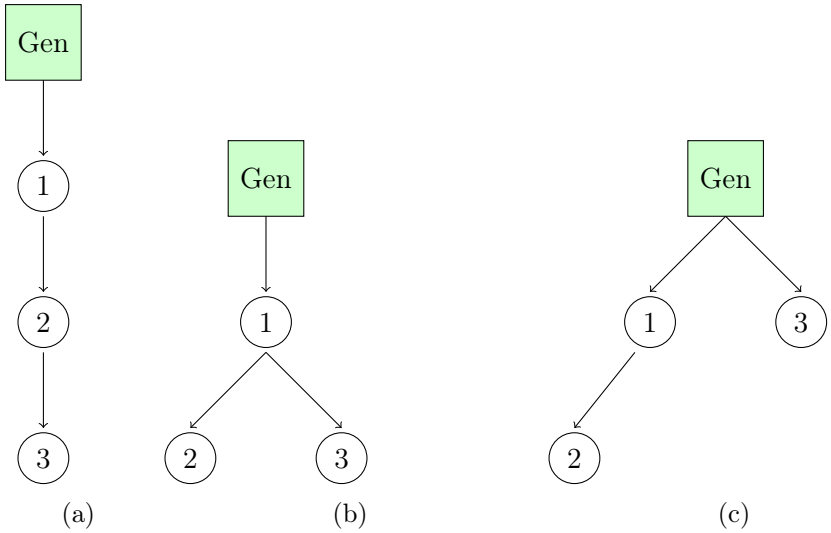


Figure 4.2: Examples of G_3 with varying structures.

For simplicity, we denote $e_{j,i,(r[1],r[2])}$ as $e_{j,r[1]}$ since for this example $k = 1$. The probability of some of the configurations of G_3 in Figure 4.2a can be written as

$$\begin{aligned} &\mathbb{P}G_3 = \text{Figure 4.2a} = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 1, e_{3,2} = 1\right) , \\ &\mathbb{P}G_3 = \text{Figure 4.2b} = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 1, e_{3,2} = 0\right) , \text{ and} \\ &\mathbb{P}G_3 = \text{Figure 4.2c} = \mathbb{P}\left(e_{2,1} = 1, e_{3,1} = 0\right) . \end{aligned}$$

Note that for the event in Figure 4.2c, it does not matter whether node $m_{(3,1)}$ has received block $(2, 1)$ or not, as the parent of that block is missing in $m_{(3,1)}$'s local tree. Block $(2, 1)$ is therefore not included in the local tree of node $m_{(3,1)}$ at that point in time.

4.4.1 Main result

Under any local attachment protocol $\mathcal{C}$ and any block arrival distribution, the event that $E_{C,t,g} = \{G_t = g\}$ depends on the random choices of proposers and polled nodes, $\{m^{(l)}_{(j,i)} : j \in [t], i \in [k], l \in [\ell]\}$, and the messages received at those respective nodes, $\{e_{j,i,r} : j \in [t], i \in [k], r[1] \in [j-1], r[2] \in [k]\}$, and some additional outside randomness on the network delay and the block arrival time. The following theorem characterizes the distribution of G_t on the system parameters $t, \Delta, \ell, \tilde{\Delta}$ for a general local attachment protocol $\mathcal{C}$ (including the longest chain and GHOST protocols). We provide a proof in Section 4.7.3.

Theorem 7. *For any local attachment protocol $\mathcal{C}$ and any inter-block arrival distribution, define random variable $\tilde{G}_t$ which takes values in the set of all possible structures of tree G_t such that* [2]

$$\mathbb{P}(\tilde{G}_t = g) \triangleq \mathbb{E}\Big[\mathbb{1}(E_{C,t,g}) \big| \{m^{(l)}_{(j,i)}\}_{j\in[t],i\in[k],l\in[\ell]} \text{ are distinct}\Big]. \tag{4.5}$$

We have the following results:

(a) *There exists a function F independent of all the parameters in the model such that for any possible tree structure g,*

$$\mathbb{P}(\tilde{G}_t = g) = F\Big(\frac{\Delta}{\ell}, \tilde{\Delta}, g, \mathcal{C}\Big). \tag{4.6}$$

(b) *The total variation distance between the distribution of G_t and $\tilde{G}_t$ is upper bounded:*

$$\mathsf{TV}\big(P_{G_t}, P_{\tilde{G}_t}\big) \leq \frac{(\ell k t)^2}{2n}. \tag{4.7}$$

In the definition in Eq. (4.5), we condition on the event that all proposers and polled nodes are distinct. This conditioning ensures that all received blocks $e_{j,i,l,r}$'s at those nodes are independent over time j. This in turn allows us to capture the precise effect of ℓ in the main result in Eq. (4.6). Further, the bound in Eq. (4.7) implies that such conditioning is not too far from the actual evolution of the blockchains, as long as the number of nodes are large enough: $n \gg (\ell k t)^2$. In practice, n need not be so large, as we show in Figure 4.3. Even with $n = 10,000 < (\ell k t)^2 = 160,000$ for 4-polling, the experiments support the predictions of Theorem 7.

The main message of the above theorem is that ℓ-Barracuda effectively reduces the network delay by a factor of ℓ. For *any* local attachment protocol and *any* block arrival process, up to a total variation distance of $(\ell k t)^2/n$, the distribution of the evolution of the blocktree with ℓ-Barracuda is the same as the distribution of the evolution of the blocktree with no polling, but with a network that is ℓ times faster. We confirm this in numerical experiments (plotted in Figure 4.3), for a choice of $\tilde{\Delta} = 0$, $n = 10,000$, $k = 1$, $t = 100$, $\gamma(t) = t$, and the longest chain fork choice rule. In the inset we show the same results, but scaled the x-axis as Δ/ℓ. As predicted by Theorem 7, the curves converge to a single curve, and are indistinguishable from one another. We used the network model from Section 4.2.2.

Without polling, the throughput degrades quickly as the network delay increases. This becomes critical as we try to scale up PoS systems; blocks should be generated more frequently, pushing

[2] The random variable $\tilde{G}_t$ is well defined, since the protocol $\mathcal{C}$ is assumed not to depend the identity of the proposer of each block. Hence, the conditional expectation is identical conditioned on each specific $\{m^{(l)}_{(j,i)} : j \in [t], i \in [k], l \in [\ell]\}$ whenever all $tk\ell$ nodes in it are distinct.

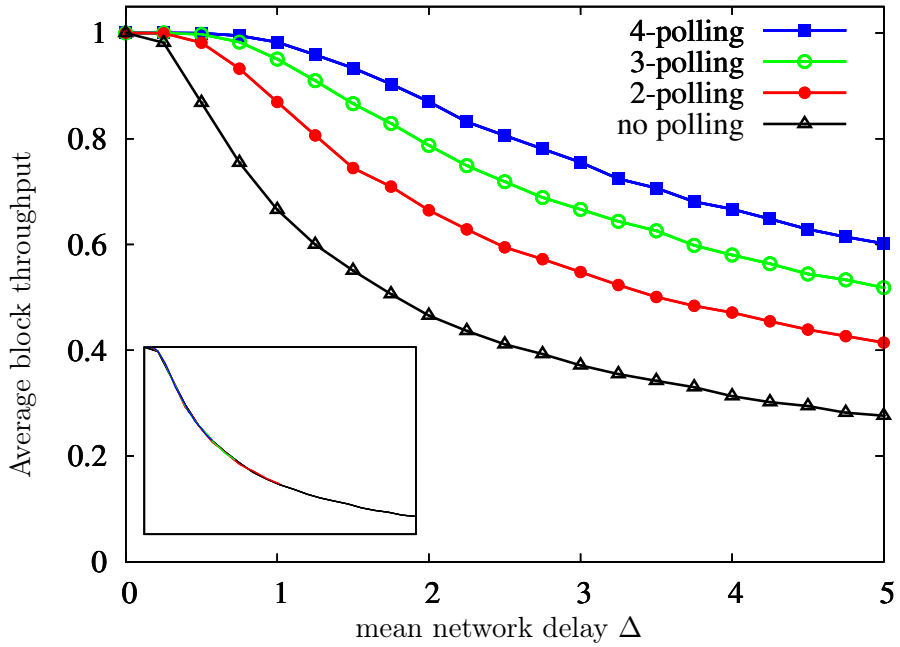


Figure 4.3: Comparing the average block throughput for various choices of ℓ confirms the theoretical prediction that ℓ-Barracuda effectively speeds up the network by a factor of ℓ; all curves are indistinguishable when x-axis is scaled as Δ/ℓ as shown in the inset.

network infrastructure to its limits. With polling, we can achieve an effective speedup of the network without investing resources on hardware upgrades. Note that in this figure, we are comparing the average block throughput, which is the main property of interest. We make this connection between the throughput and ℓ precise in the following. Define $L_{\mathsf{Chain}}(G_t)$ to be the length of the longest chain in G_t excluding the Genesis block. Throughput is defined as $\mathbb{E}[L_{\mathsf{Chain}}(G_t)]/t$. We have the following Corollary of Theorem 7.

Corollary 1. *There exists a function $L(\Delta/\ell, \tilde{\Delta}, \mathcal{C})$ independent of all the parameters in the model such that*

$$\left| \mathbb{E}[L_{\mathsf{Chain}}(G_t)] - \mathbb{E}\Big[L\Big(\frac{\Delta}{\ell}, \tilde{\Delta}, \mathcal{C}\Big)\Big] \right| \leq \frac{t(\ell k t)^2}{2n}. \tag{4.8}$$

In other words, in the regime that $n \gg t^3(k\ell)^2$, the expectation of the length of the longest chain depends on the delay parameter Δ and the polling parameter ℓ only through their ratio Δ/ℓ. Hence, the block throughput enjoys the same polling gain, as the distribution of the resulting block trees.

4.4.2 Connections to balls-in-bins example

In this section, we give a brief explanation of the balls-in-bins problem and the power of two choices in load balancing. We then make a concrete connection between the blockchain problem and the power of ℓ-polling in information balancing.

In the classical balls-in-bins example, we have t balls and t bins, and we sequentially throw each ball into a uniformly randomly selected bin. Then, the maximum loaded bin has load (i.e. number of balls in that bin) scaling as $\Theta(\log t/\log\log t)$ [127]. The result of *power of two choices* states that

if every time we select ℓ ($\ell \geq 2$) bins uniformly at random and throw the ball into the *least* loaded bin, the maximum load enjoys an near-*exponential* reduction to $\Theta(\log\log t/\log \ell)$ [127].

Our polling idea is inspired by this power of two choices in load balancing. We make this connection gradually more concrete in the following. First, consider the case when the underlying network is extremely slow such that no broadcast of the blocks is received. When there is no polling, each node is only aware of its local blockchain consisting of only those blocks it generated. There is a one-to-one correspondence to the balls-in-bins setting, as blocks (balls) arriving at each node (bin) build up a load (local blockchain). When there are t nodes and t blocks, then it trivially follows that the length of the longest chain scales as $\Theta(\log t/\log\log t)$, when there is no polling.

The main departure is that in blockchains, the goal is to maximize the length of the longest chain (maximum load). This leads to the following fundamental question in the balls-in-bins problem, which has not been solved, to the best of our knowledge. If we throw the ball into the *most* loaded bin among ℓ randomly chosen bins at each step, how does the maximum load scale with t and ℓ? That is, if one wanted to maximize the maximum load, leading to load unbalancing, how much gain does the power of ℓ choices give? We give a precise answer in the following.

Theorem 8. *Given t empty bins and t balls, we sequentially allocate balls to bins as follows. For each ball, we select uniformly at random ℓ bins, and put the ball into the maximally-loaded bin among the ℓ chosen ones. Then, the maximum load of the t bins after the placement of all t balls is at most*

$$C \cdot \ell \cdot \frac{\log t}{\log\log t} \tag{4.9}$$

with probability at least $1 - \frac{1}{t}$, where $C > 0$ is a universal constant.

We provide a proof in Section 4.7.4. This shows that the gain of ℓ-polling in maximizing the maximum load is linear in ℓ. Even though this is not as dramatic as the exponential gain of the load balancing case, this gives a precise characterization of the gain in the throughput of ℓ-Barracuda in blockchains when $\Delta \gg 1$. This is under a slightly modified protocol where the polling happens in a bidirectional manner, such that the local tree and the newly appended block of the proposer are also sent to the polled nodes.

For moderate to small Δ regime, which is the operating regime of real systems, blocktree evolution is connected to a generalization of the balls-in-bins model. Now, it is as if the balls are copied and broadcast to all other bins over a communication network. This is where the intuitive connection to balls-and-bins stop, as we are storing the information in a specific data structure that we call blocktrees. However, we borrow the terminology from 'load balancing', and refer to the effect of polling as 'information balancing', even though load balancing refers to *minimizing* the maximum load, whereas information balancing refers to *maximizing* the maximum load (longest chain) by balancing the information throughout the nodes using polling.

4.5 System and implementation issues

We empirically verify the robustness of our proposed protocol under various issues that might come up in a practical implementation of ℓ-Barracuda. Our experiment consists of n nodes connected via a network which emulates the end to end delay as an exponential distribution; this model is inspired by the measurements of the Bitcoin P2P network made in [59].

Each of the n nodes maintains a local blocktree which is a subset of the global blocktree. We use a deterministic block arrival process with $\gamma(t) = t$, i.e. we assume a unit block arrival time which is also termed as an epoch in this section. This represents an upper bound on block arrivals in

real-world PoS systems, where blocks can only arrive at fixed time intervals. At the start of arrival t, k proposers are chosen at random and each of these proposers proposes a block.

When there is no polling, each proposer chooses the most eligible block from its blocktree to be a parent to the block it is proposing, based on the fork choice rule. In the case of ℓ-Barracuda, the proposer sends a pull message to $\ell - 1$ randomly chosen nodes, and these nodes send their block tree back to the proposer. The proposer receives the block trees from the polled nodes after a delay $\tilde{\Delta}$, and updates her local blocktree by taking the union of all received blocktrees. The same fork choice rule is applied to decide the parent to the newly generated block. In all our experiments we use the Nakamoto longest chain fork choice rule. We ran our experiments for $T = 100$ time epochs on a network with $n = 10,000$ nodes with $k = 1$.

4.5.1 Effect of polling delay

In reality, there is delay between initializing a poll request and receiving the blocktree information. We expect polling delay to be smaller than the delay of the P2P relay network because polling communication is point-to-point rather than occurring through the P2P relay network. To understand the effects of polling delay, we ran simulations in which a proposer polls $\ell - 1$ nodes at the time of proposal, and each piece of polled information arrives after time $\tilde{\Delta}_1$, $\tilde{\Delta}_2$, .., $\tilde{\Delta}_{\ell-1} \sim \mathsf{Exp}(\frac{1}{0.1\,\Delta})$. The proposer determines the pointer of the new block when all polled messages are received.

Figure 4.4 shows the effect of such polling delay, as measured by $\Delta_{0.8}(\ell)$, the largest delay Δ that achieves a block throughput of at least 0.8 under ℓ-Barracuda. More precisely,

$$\Delta_{0.8}(\ell) = \max\left\{\Delta \,:\, \lim_{t\to\infty} \frac{\mathbb{E}[L(G_t)]}{t} \geq 0.8\right\}.$$

Under this model, polling more nodes means waiting for more responses; the gains of polling hence saturate for large enough ℓ, and there is an appropriate practical choice of ℓ that depends on the interplay between the P2P network speed and the polling delay.

In practice, there is a strategy to get a large polling gain, even with delays: the proposer polls a large number of nodes, but only waits a fixed amount of time before making a decision. Under this protocol, polling more nodes can only help; the only cost of polling is the communication cost. The results of our experiments under this protocol are illustrated in Figure 4.5 ('no setup delay' curve).

This implies a gap in our model, which does not fully account for the practical cost of polling. In order to account for polling costs, we make the model more realistic by assigning a small and constant delay of 0.01Δ to set up a connection with a polling node, and assume that the connection setup occurs sequentially for $\ell - 1$ nodes. The proposer follows the same strategy as above: waiting for a fixed amount of time before making the decision. We see that under such model, there is a finite optimal ℓ as shown in Figure 4.5.

4.5.2 Heterogeneous networks

The theoretical and experimental evidence of the benefits of ℓ-Barracuda have so far been demonstrated in the context of a homogeneous network: all the nodes in the network have the same bandwidth and processing speeds. Further the individual variation in end-to-end delay due to network traffic is captured by statistically identical exponential random variables. In practice, heterogeneity is natural (some nodes have stronger network capabilities); we model this by clustering the nodes into h different groups, based on their average network speed. The speed of a connection between two nodes is determined by the speed of the slower node. We compare the performance of ℓ-Barracuda with that of no polling (which has worse performance and serves as a lower bound).

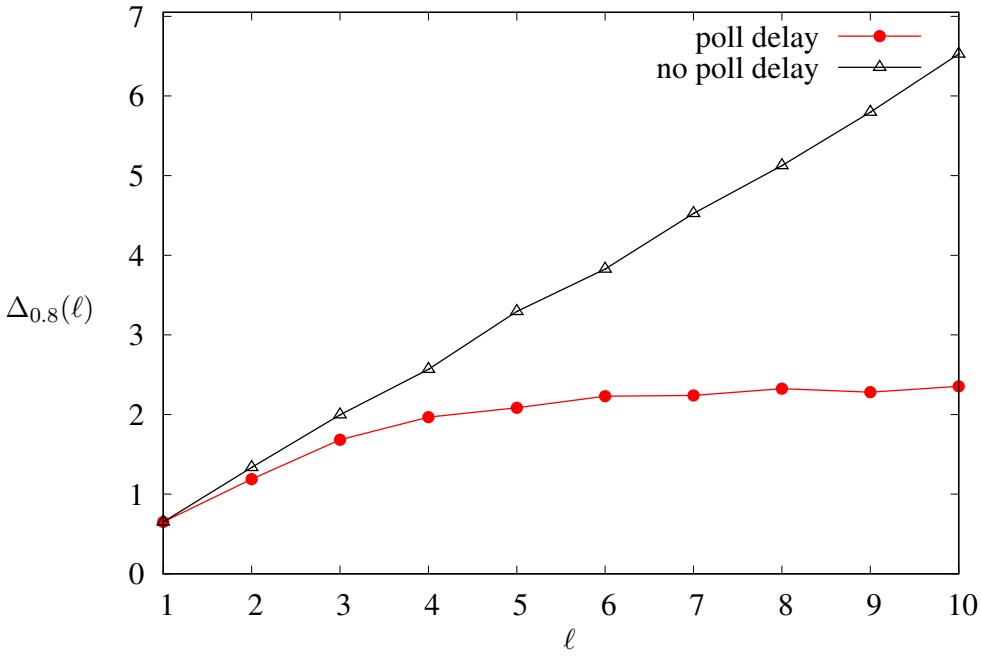


Figure 4.4: $\Delta_{0.8}(\ell)$ captures the highest delay Δ that achieves a desired block throughput of 0.8 under ℓ-Barracuda. With a polling delay of $\mathsf{Exp}(1/(0.1\Delta))$, the performance saturates after $\ell = 6$ and eventually deteriorates at large ℓ.

We follow the following uniform polling strategy: Let the delay Δ of a node be a part of the set $\mathcal{D} = \{\Delta_1, \Delta_2, .., \Delta_h\}$; a node's delay is defined as follows: the average delay of transmitting a block across the P2P network from node with delay Δ_i to a node with delay Δ_j is $\max(\Delta_i, \Delta_j)$ $\forall i \in [h]$.

In Figure 4.6, we show the performance of a heterogeneous network with $h = 2$: half of the nodes have delay Δ and the other half have delay 5Δ. Every node has the same proposer election probability. ℓ-Barracuda gives a gain in the network throughput in line with the prediction of Theorem 7, even when the underlying network is heterogeneous.

4.5.3 Polling partial blocktrees

The polling studied in this paper requires syncing of the complete local blocktree, which is redundant and unnecessarily wastes network resources. For efficient bandwidth usage, we propose (ℓ, b)-polling, where the polled nodes only send the blocks that were generated between times $t-1$ and $t-b$. In Figure 4.7, we compare the performance of 2-polling and (2,1),(2,3),(2,5)-polling. The experiments suggest that comparable performance can be achieved with a choice of b that is small, and that choice increases with network delay.

4.5.4 Incentive Structure

To ensure timely response from polled nodes, we propose appropriate incentive mechanims, motivated by those used in BitTorrent. When BitTorrent nodes explore their neighbors to get new information, a reputation system is maintained, where nodes that upload have higher reputation, thus allowing for faster downloads. Incentive mechanisms for BitTorrent-like P2P networks have been analyzed to achieve Nash equilibrium as shown by [146]. We propose a reputation system for blockchain polling,

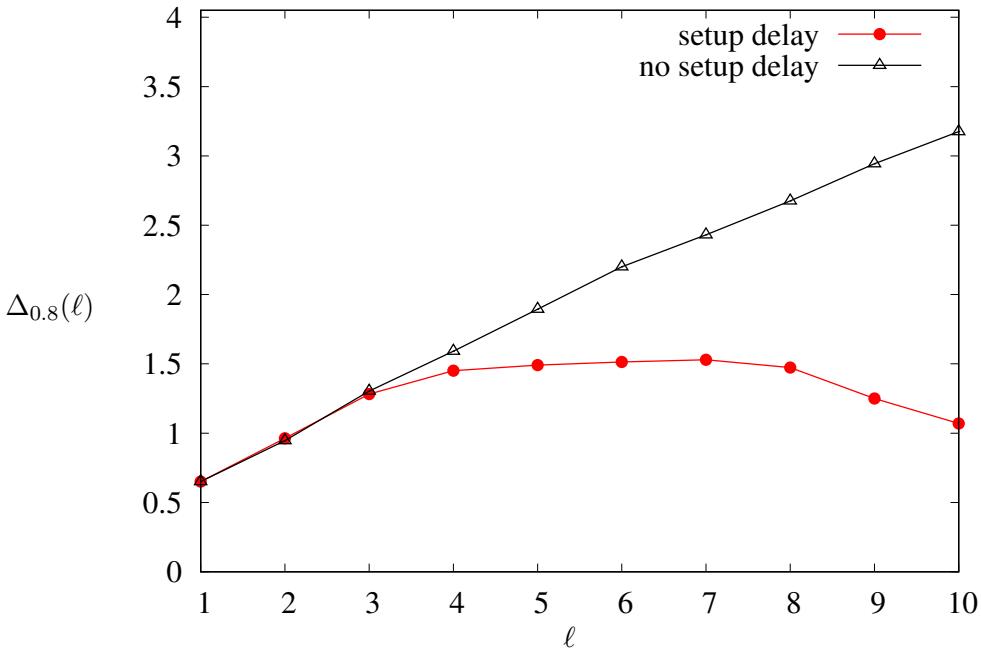


Figure 4.5: Average delay $\Delta_{0.8}(\ell)$ that achieves desired block throughput 0.8, when ℓ-Barracuda is used. We assume there is a polling delay of $\mathsf{Exp}(1/(0.1\Delta))$ but the proposer waits exactly $\tilde{\Delta} = 0.1\Delta$ time before proposing. When there is no setup delay, polling more nodes is always better. Otherwise, we see an optimal ℓ, which depends on all system parameters.

where a node replies to a poll request only from a proposer with a higher reputation (tit-for-tat). A higher reputation is maintained by honestly responding to a polling request.

Another incentive mechanism involves the proposer paying polling fees to the the polled nodes. The fees can be in the form of a conditional payment, conditioned on the proposed block being finalized or reaching a depth of m blocks in the blockchain. Let P_{c} be the probability that block c is finalized conditioned on the polled node sending its blocktree, and P_{nc} be the probability that block is finalized conditioned on the polled node sending nothing. Then the polled node should send its complete blocktree if:

$$R(P_{\mathrm{c}} - P_{\mathrm{nc}}) \quad > \quad C$$

where R is the polling fees/reward and C is the cost of sending blocktree to the proposer. Note that sending the blocktree only involves sending the difference of the blocktree of the polled node and the proposer, hence the cost C is relatively small. Sufficiently large fee R ensures that the nodes are incentivized to respond.

4.5.5 Security Implications

Blockchains are expected to operate in *permissionless* conditions, so a fraction of the participants may deviate from the proposed protocol with explicit malicious intent (of causing harm to the key performance metrics). It is natural to explore potential security vulnerabilities exposed by the ℓ-Barracuda protocol proposed in this paper. Since push and poll operations are different network primitives and since the poll operation is performed by proposer nodes, nodes when polled ascertain who the proposer of a new block is. Such nodes could use this information perhaps initiate a denial of service attack on the proposer (or launch a bribery attack involving corrupting the proposer

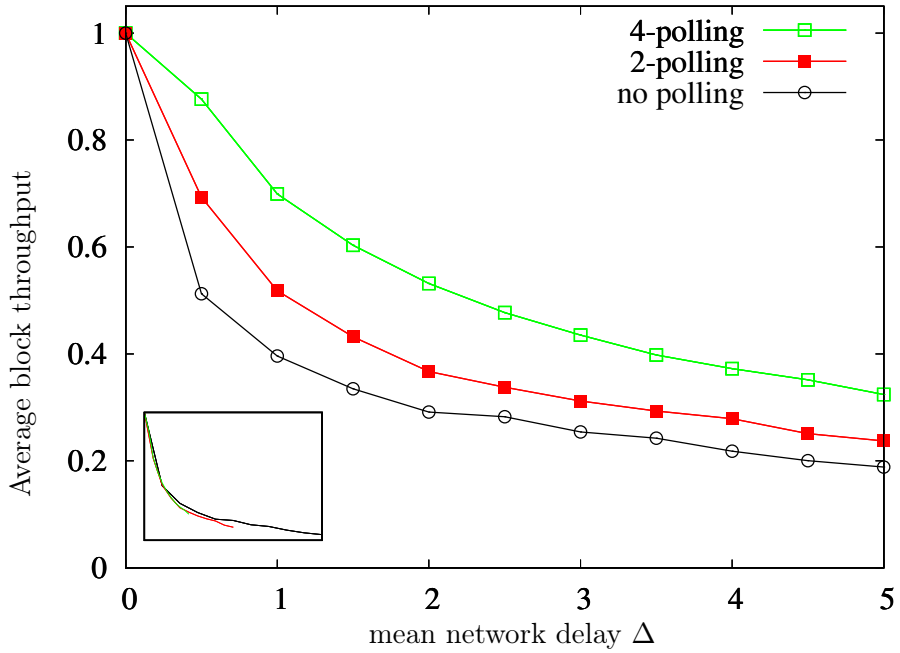


Figure 4.6: In a heterogeneous network, we enjoy the same polling gain as in a homogeneous network. Heterogeneous ℓ-Barracuda provides a speedup of the network by a factor of about ℓ, as shown in the inset where the x-axis is scaled by Δ/ℓ.

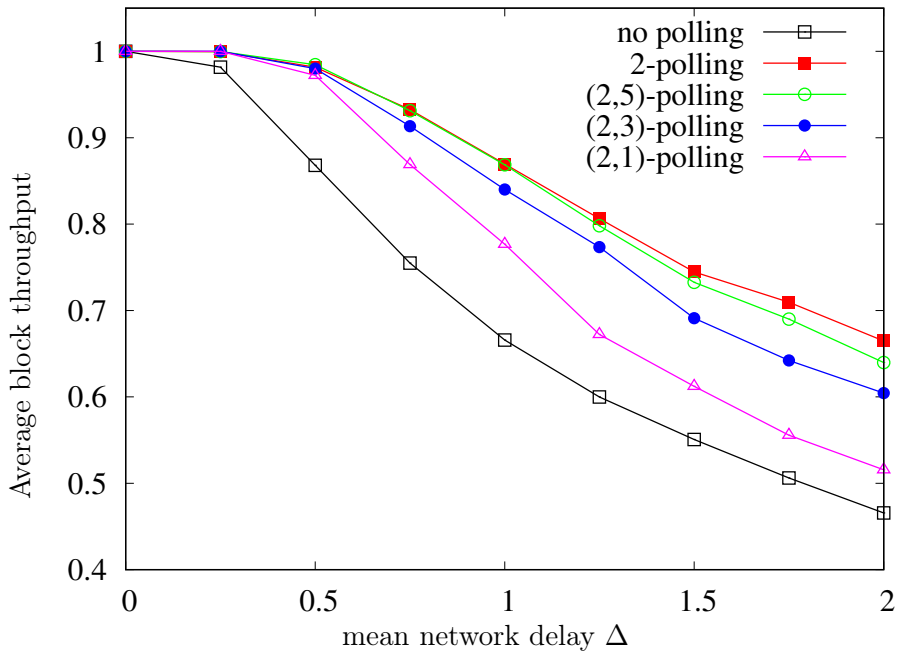


Figure 4.7: Sending the latest 5 blocks when polled is sufficient to achieve comparable performance as sending the entire local blocktree, while occupying significantly less network bandwidth.

of the new block) – these possibilities are similar to (but more muted than) the attacks on the

class of consensus protocols summarized under the item on "reducing proposer diversity" (example: Bitcoin-NG) and the vulnerabilities are no more than faced by this class of blockchain algorithms. However, a simple modification to the ℓ-Barracuda protocol nullifies even this relatively minor vulnerability.

Consider the following network protocol design symmetric with respect to polling: we replace push and poll network primitives by a single block tree information exchange primitive that we call "information-sync". Each connection/edge information-sync involves exchanging block tree information in a symmetric way between the pair of nodes. A new sync starts as soon as the previous sync finishes. When a proposer is elected, it starts $\ell - 1$ new sync connections and proposes a block after completing sync from these $\ell - 1$ connections. As we can see those $\ell - 1$ nodes will consider this polling request as just another node asking for a sync, allowing the poll request to hide within the symmetrical structure. We tested this new protocol for the following parameters: $n = 500$ and $T = 100$ for a 4-regular graph with $\mathsf{Exp}(\Delta)$ edge synchronization delays. The performance is shown in Figure 4.8, where we see that 5-polling offers a significant gains compared to no polling. Note that the overall gains are somewhat muted as compared to the polling gains on the pure ℓ-Barracuda protocol setting, as expected: the symmetric network protocol already includes some polling due to the nature of the information-sync primitive.

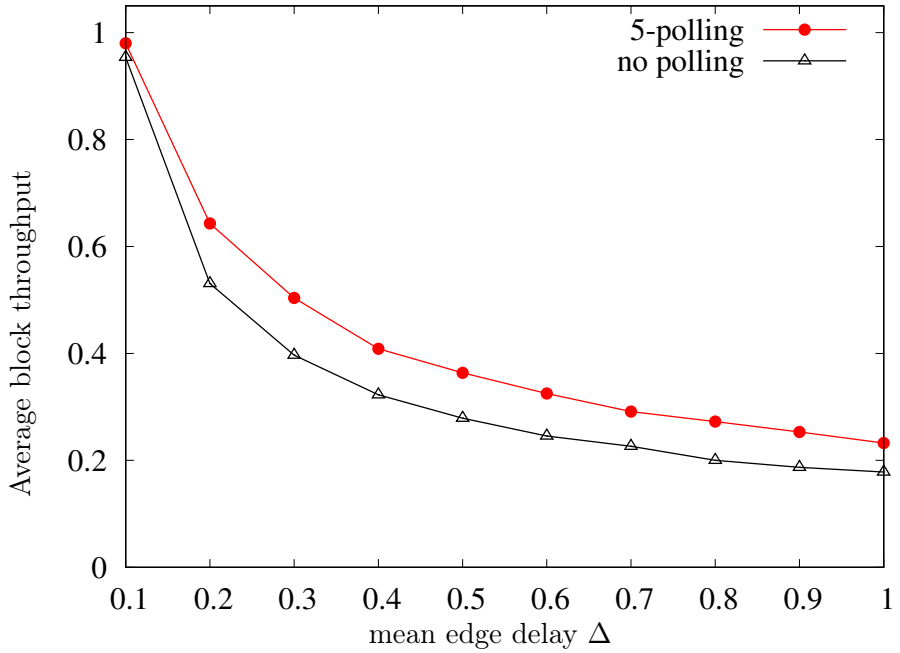


Figure 4.8: 5-polling shows gain over no polling in a homogeneous information sync network

Polling also helps when the network is heterogeneous, as shown in Figure 4.9, which uses the same node delay distribution as the heterogeneous exponential model discussed in section 4.2. A node's delay is defined as follows: the average edge synchronization delay between nodes with delay Δ_i and Δ_j is $\max(\Delta_i, \Delta_j)$ $\forall i \in [h]$. Here both the slower nodes and the faster nodes poll the same number of nodes.

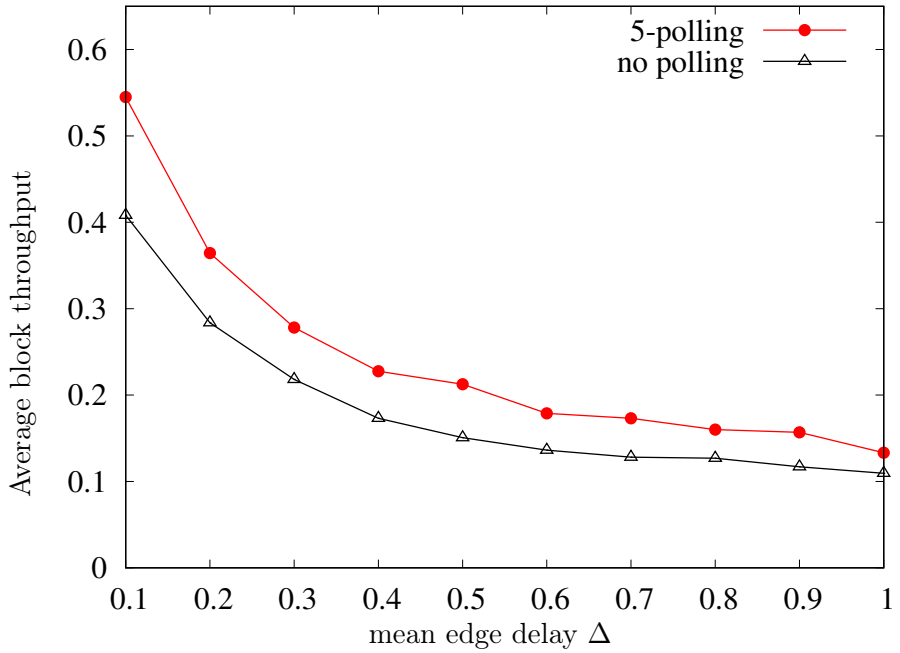


Figure 4.9: 5-polling shows gain over no polling in a heterogeneous information sync network

4.6 Relation to Prism

In this chapter, we propose ℓ-Barracuda as a technique for improving block throughput in proof-of-stake cryptocurrencies. We show that for small ℓ, ℓ-Barracuda has the same effect on block throughput as if the mean network delay were reduced by a factor of ℓ. This is a simple, lightweight method of improving throughput without needing to substantially alter either the underlying consensus protocol or the properties of the network. In particular, this result holds for *any* local attachment protocol, including the longest-chain fork choice rule. Since each individual blocktree in Prism uses a longest-chain fork choice rule, Barracuda can easily be integrated with Prism. Namely, each proposer, upon being elected, would poll ℓ neighbors for their views of the $m+1$ blocktrees, where m is the number of voter trees in Prism. Based on this polled information, the proposer would update its local view and propose a new block.

4.7 Proofs of the main results

We provide the proofs of the main results in this section.

4.7.1 Proof of Theorem 5

Since $k = 1$, we denote the proposer of block j as m_j. Thus the arrival time of the block j to node m is given by

$$R_{j,m} = \begin{cases} j & \text{if } m = m_j \\ j + B_{j,m} & \text{if } m \neq m_j \end{cases}.$$

Here $B_{j,m} \sim \mathsf{Exp}(1/\Delta)$ and they are mutually independent random variables for $1 \le j \le t, 1 \le m \le n$. Define the event

$$e_{j,m,r} = \begin{cases} 1 & \text{when block } j \text{ is proposed, node } m \text{ has received block } r \\ 0 & \text{otherwise} \end{cases}$$

Thus whenever node m_t is chosen as a proposer for block t, the set of events $\{e_{t,m_t,r} : 1 \le r \le t-1\}$ determine the length of the longest chain $L_t \triangleq L_{\mathsf{Chain}}(G_t)$. In particular, if we denote the blocks that are part of a longest chain $\mathsf{Chain}(G_{t-1})$ as $j_1, \ldots, j_{L_{t-1}}$, we have that

$$L_t = L_{t-1} + 1, \quad \text{if } \prod_{r=1}^{L_{t-1}} e_{t,m_t,j_r} = 1.$$

In other words, had the node m_t received all the blocks that were part of a longest chain at time $t-1$, the length L_t would exceed L_{t-1} by 1. Since $L_t \ge L_{t-1}$ with probability 1, we obtain that

$$\mathbb{E}[L_t] \ge \mathbb{E}[L_{t-1}] + \mathbb{P}e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}].$$

Fix any $m_t \in [n]$. If m_t had not been a proposer for any of the blocks j_r, we get that

$$\mathbb{P}e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}] = \prod_{r=1}^{L_{t-1}} (1 - x^{t-j_r}), \quad x = e^{\frac{-1}{\Delta}}$$

Had m_t been a proposer before, clearly the above probability is still a lower bound on $\mathbb{P}e_{t,m_t,j_r} = 1, \forall r \in [L_{t-1}]$. Thus

$$\mathbb{E}[L_t] \ge \mathbb{E}[L_{t-1}] + \prod_{r=1}^{L_{t-1}} (1 - x^{t-j_r}) \ge \mathbb{E}[L_{t-1}] + \prod_{i=1}^{\infty} (1 - x^i).$$

Applying the above inequality recursively, we obtain that

$$\mathbb{E}[L_{\mathsf{Chain}}(G_t)] \ge (\prod_{i=1}^{\infty} (1 - x^i)) \cdot t.$$

Thus it suffices to lower bound the Euler function $\phi(x) \triangleq \prod_{i=1}^{\infty}(1 - x^i)$. Using the inequality that $\frac{x}{1+x} \le \log(1+x), x \ge -1$, we obtain that

$$\begin{aligned} \log \phi(x) &= \sum_{i \ge 1} \log(1 - x^i) = \sum_{i \ge 1} \frac{-x^i}{1 - x^i} \\ &\ge \sum_{i \ge 1} \frac{-x^i}{1 - x} = \frac{-x}{1 - x} \sum_{i \ge 0} x^i = \frac{-x}{(1-x)^2}. \end{aligned}$$

Thus $\phi(x) \ge \exp\left(\frac{-x}{(1-x)^2}\right)$.

4.7.2 Proof of Theorem 6

We apply Theorem 7 with general $\ell \geq 1$ and then specialize it to $\ell = 1$ to obtain the theorem statement. Denote the chain as g, and $e_{j,i,r[1],r[2]}$ as $e_{j,r[1]}$ since here $k = 1$. The event $E_{C,t,g}$ can be written as

$$\begin{aligned} E_{C,t,g} = \mathbb{1}(e_{2,1} = 1) \cdot \mathbb{1}(e_{3,1} = 1, e_{3,2} = 1) \cdot \\ \ldots \cdot \mathbb{1}(e_{t,1} = 1, e_{t,2} = 1, \ldots, e_{t,t-1} = 1). \end{aligned} \tag{4.10}$$

Let $\tilde{E}$ denote the event that every node has proposed or been polled at most once. Conditioned on $\tilde{E}$, and defining $\alpha \triangleq e^{\tilde{\Delta} l/\Delta}$, we have

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] = \prod_{j=2}^{t} \mathbb{E}[\mathbb{1}(e_{j,1} = 1, e_{j,2} = 1, \ldots, e_{j,j-1} = 1)|\tilde{E}] \tag{4.11}$$

$$= \prod_{j=2}^{t} \prod_{m=1}^{j-1} (1 - e^{\tilde{\Delta} l/\Delta} e^{-m\ell/\Delta}) \tag{4.12}$$

$$= \prod_{j=1}^{t-1} (1 - \alpha e^{-j\ell/\Delta})^{t-j} \tag{4.13}$$

$$\leq (1 - \alpha e^{-\ell/\Delta})^{t-1}. \tag{4.14}$$

We now claim that if $\ell \geq \frac{\Delta\left(\ln t - \ln\ln\frac{1}{\delta}\right)}{1-\tilde{\Delta}}$, we have $\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \geq \delta - o(1)$. Let $c = \ln\frac{1}{\delta}$.

Indeed, in this case, we have $\alpha e^{-\ell/\Delta} \leq \frac{\ln\frac{1}{\delta}}{t}$. Hence,

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \geq \prod_{j=1}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{t-j} \tag{4.15}$$

$$= \left(1 - \frac{c}{t}\right)^{t\frac{t-1}{t}} \prod_{j=2}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{t-j} \tag{4.16}$$

$$\overset{(a)}{\geq} e^{-c} - o(1) \tag{4.17}$$

$$= \delta - o(1), \tag{4.18}$$

where (a) follows from 4.7.1 and the fact that $\lim_{t\to\infty}(1 - c/t)^{t-1} = e^{-c}$. Conversely, we show that if $\ell \leq \frac{\Delta\left(\ln t - \ln\ln\frac{1}{\delta}\right)}{1-\tilde{\Delta}}$, then $\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \leq \delta + o(1)$.

Indeed, in this case we have $\alpha e^{-\ell/\Delta} \geq \frac{\ln\frac{1}{\delta}}{t}$, and

$$\mathbb{E}[\mathbb{1}(E_{C,t,g})|\tilde{E}] \leq (1 - c/t)^{t-1} \tag{4.19}$$

$$= e^{-c} + o(1) \tag{4.20}$$

$$= \delta + o(1). \tag{4.21}$$

Lemma 4.7.1. *Let $c > 0$ be fixed. Then we have that*

$$\lim_{n\to\infty} \sum_{k=2}^{n-1} (n-k) \log(1 - \frac{c^k}{n^k}) = 0.$$

Proof. Define $f_n(\cdot) : \mathbb{N} \to \mathbb{R}$ as

$$f_n(k) = (n-k)\log(1-\frac{c^k}{n^k})\mathbb{1}\{2 \le k \le n-1\}.$$

For each fixed $k \in \mathbb{N}$, we have that $\lim_{n\to\infty} f_n(k) = 0$. Our goal is to show that $\lim_{n\to\infty} \int_{\mathbb{N}} f_n(k)d\mu(k) = 0$ where $\mu(\cdot)$ is the counting measure on $\mathbb{N}$. In view of dominated convergence theorem, hence it suffices to show that there exists a $g : \mathbb{N} \to \mathbb{R}$ such that

$$|f_n(k)| \le g(k), \quad k \in \mathbb{N} \text{ and } \int g d\mu < \infty.$$

Note that for $2 \le k \le n-1$,

$$|f_n(k)| = (n-k)|\log(1-\frac{c^k}{n^k})| \overset{(a)}{\le} (n-k)\frac{\frac{c^k}{n^k}}{1-\frac{c^k}{n^k}} \le \frac{nc^k}{n^k-c^k},$$

where (a) follows from the fact that $|\log(1-x)| \le \frac{x}{1-x}$ for $x \in [0,1]$. Let $n_0 \in \mathbb{N}$ be such that $n_0^k \ge 2c^k$. Hence, for $n \ge n_0$, we have

$$|f_n(k)| \le \left(\frac{2}{c}\right)\left(\frac{c}{n}\right)^{k-1} \le \left(\frac{2}{c}\right)\left(\frac{c}{n_0}\right)^{k-1} \triangleq g(k).$$

Clearly $\int_{\mathbb{N}} g(k)d\mu(k) < \infty$. The claim follows. □

4.7.3 Proof of Theorem 7

Part (1). One key observation is that, if every node has only been polled or proposed at most once, i.e,, the set $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ contains $tk\ell$ distinct nodes, then conditioned on this specific sequence $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$, all the random variables $\{e_{j,i,l,r} : j \in [t], i \in [k], l \in [\ell], r[1] \in [j-1], r[2] \in [k]\}$ are mutually independent. Furthermore, conditioned on this specific sequence, we have

$$\mathbb{E}[e_{j,i,l,r}|\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}, \{\gamma(i)\}_{i=1}^t] \tag{4.22}$$

$$= \ 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})/\Delta}, \tag{4.23}$$

for all r such that $r[1] \in [j-1], r[2] \in [k]$.

Denote the event of $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ are distinct as $\tilde{E}$. It follows from the definition of the local attachment protocol $\mathcal{C}$ that

$$\mathbb{E}[e_{j,i,l,r}|\tilde{E}, , \{\gamma(i)\}_{i=1}^t] = 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})/\Delta} \tag{4.24}$$

for all r such that $r[1] \in [j-1], r[2] \in [k]$.

Note that the event $E_{C,t,g} = \{G_t = g\}$ only depends on $\{m_{(j,i)}^{(l)} : j \in [t], i \in [k], l \in [\ell]\}$ and $\{e_{j,i,r} : j \in [t], i \in [k], r[1] \in [j-1], r[2] \in [k]\}$ plus some additional outside randomness. Since

$$e_{j,i,r} = 1 \Leftrightarrow \sum_{l\in[\ell]} e_{j,i,l,r} \ge 1, \tag{4.25}$$

it follows from the independence of $e_{j,i,l,r}$ and equation (4.22) that

$$\mathbb{E}[e_{j,i,r}|\tilde{E},\{\gamma(i)\}_{i=1}^{t}] = 1 - e^{-(\gamma(j)-\gamma(r[1])-\tilde{\Delta})\ell/\Delta} \tag{4.26}$$

all r such that $r[1] \in [j-1], r[2] \in [k]$.

Hence, we have

$$\mathbb{P}(\tilde{G}_t = g) = \mathbb{E}\left[\mathbb{1}(E_{C,t,g})|\tilde{E},\{\gamma(i)\}_{i=1}^{t}\right] \tag{4.27}$$

$$= F(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, g, \mathcal{C}). \tag{4.28}$$

Now we show the second part of Theorem 7. Denote by $A = \{g_1, g_2, \ldots, g_A\}$ any collection of distinct tree structures that G_t may take values in. Then, we have

$$\begin{aligned}
&\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) \\
&= \mathbb{E}\left[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})\middle|\{\gamma(i)\}_{i=1}^{t}\right] &(4.29)\\
&= \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t})\mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^{t}] \\
&\quad + (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}))\mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^{t}] &(4.30)\\
&= \mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^{t}] \\
&\quad + (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t})) \\
&\quad \times \left(\mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^{t}] - \mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^{t}]\right) &(4.31)\\
&= \mathbb{P}(\tilde{G}_t \in A) \\
&\quad + (1-\mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t})) \\
&\quad \times \left(\mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E}^c,\{\gamma(i)\}_{i=1}^{t}] - \mathbb{E}[\sum_{i=1}^{A}\mathbb{1}(E_{C,t,g_i})|\tilde{E},\{\gamma(i)\}_{i=1}^{t}]\right). &(4.32)
\end{aligned}$$

It follows from the birthday paradox computation [127, Pg. 92] that

$$1 - \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}) \leq \frac{kt\ell(kt\ell-1)}{2n}, \tag{4.33}$$

Hence, we have shown that for any measurable set A that G_t or $\tilde{G}_t$ take values in, we have

$$|\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) - \mathbb{P}(\tilde{G}_t \in A)| \leq 1 - \mathbb{P}(\tilde{E}|\{\gamma(i)\}_{i=1}^{t}) \tag{4.34}$$

$$\leq \frac{kt\ell(kt\ell-1)}{2n}. \tag{4.35}$$

The result follows from the definition of the total variation distance

$$\mathsf{TV}(P_{G_t|\{\gamma(i)\}_{i=1}^{t}}, P_{\tilde{G}_t}) = \sup_{A}|\mathbb{P}(G_t \in A|\{\gamma(i)\}_{i=1}^{t}) - \mathbb{P}(\tilde{G}_t \in A)|. \tag{4.36}$$

Part (2). We note that there exists some function $L(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, \mathcal{C})$ independent of all the parameters in the model such that the expectation of the longest chain of $\tilde{G}_t$ is equal to $L(\{\gamma(i)\}_{i=1}^{t}, \frac{\Delta}{\ell}, \tilde{\Delta}, \mathcal{C})$. To obtain the final result, it suffices to use the variational representation of total variation distance:

$$\mathsf{TV}(P, Q) = \sup_{f:|f|\leq \frac{1}{2}} \mathbb{E}_P f - E_Q f, \tag{4.37}$$

and taking $f = \frac{1}{t} \cdot (L_{\mathsf{Chain}}(G_t) - t/2)$, upon noticing that the length of the longest chain in the tree G_t is at most t.

4.7.4 Proof of Theorem 8

We index all the t balls as $1, 2, \ldots, t$. Denote the load of the maximally loaded bin after the placement of all balls at L_t. We aim at upper bounding

$$\mathbb{P}\left(L_t \geq k\right). \tag{4.38}$$

for $k \geq C \cdot \ell \cdot \frac{\log t}{\log\log t}$. It follows from the union bound that

$$\begin{aligned}&\mathbb{P}\left(L_t \geq k\right)\\ &\quad\leq \sum_{j\in[t]} \sum_{k-\text{subset } (i_1,i_2,\ldots,i_k)\in[t]^k} \mathbb{P}\left(\text{Bin } j \text{ has balls } (i_1, i_2, \ldots, i_k)\right).\end{aligned}$$

Hence, it suffices to upper bound each individual term. Note that in the placement of each ball, the probability that a specific bin was selected as a potential candidate is at most $\frac{\binom{t-1}{\ell-1}}{\binom{t}{l}} = \frac{\ell}{t}$. Hence, for each k-subset $(i_1, i_2, \ldots, i_k)$, the probability that bin j contains this k-subset is at most $\left(\frac{\ell}{t}\right)^k$. Applying the union bound, we have

$$\mathbb{P}\left(L_t \geq k\right) \leq t\binom{t}{k}\left(\frac{\ell}{t}\right)^k.$$

Using the fact that $\left(\frac{n}{e}\right)^n \leq n! \leq e\sqrt{n}\left(\frac{n}{e}\right)^n$ for $n \geq 1$, we have

$$\begin{aligned}t\binom{t}{k}\left(\frac{\ell}{t}\right)^k &\leq t\frac{e\sqrt{t}t^t}{(t-k)^{t-k}k^k}\frac{\ell^k}{t^k}\\ &\leq et^{3/2}\left(\frac{t}{t-k}\right)^{t-k}\left(\frac{\ell}{k}\right)^k\\ &\leq et^{3/2}\left(1+\frac{k}{t-k}\right)^{\frac{t-k}{k}\cdot k}\left(\frac{\ell}{k}\right)^k\\ &\leq et^{3/2}\left(\frac{e\ell}{k}\right)^k.\end{aligned}$$

Setting $et^{3/2}\left(\frac{e\ell}{k}\right)^k \leq \frac{1}{t}$, the theorem is proved.

4.8 Acknowledgement

The authors thank A. Makkuva, and R. Rana for collaborations on the design of Barracuda. This chapter includes and expands on material in [67].

Chapter 5

Polyshard: Scalable Storage and Computation

Salman Avestimehr, USC
Sreeram Kannan, UW-Seattle
Pramod Viswanath, UIUC

Blockchain systems, which maintain a decentralized trusted ledger and can do finite-state computations can execute a wide range of programs in a trust-free setting. This has promised a host of new and exciting applications in various areas. However, to enable these applications, we need blockchain systems that *scale well with the number of participating nodes* n. In this chapter, we focus on two important performance metrics: *throughput* (number of transactions validated per discrete round) λ and the inherent *storage capacity* of the blockchain (measured in terms of the entropy of the total storage among all the nodes) γ. By scalable, we mean that these two performance metrics should *improve* as the number of nodes n increases.

We begin this chapter by observing how the two performance metrics λ, γ scale with n in extant blockchain designs (example: Bitcoin or Ethereum). In these traditional technologies, the key underpinning storage and computation methods involve *full replication*:

> every node must replicate all the computation and every node must store the entire blockchain.

We denote the amount of storage at a node by S and the amount of transactions that can be processed at a node (per unit discrete round) by T. Then systems based on full replication have the basic upper bound on performance:

$$\text{Upper bounds for Traditional Blockchains}: \qquad \lambda \leq T; \qquad \gamma \leq S,$$

i.e., traditional methods cannot process more throughput than T transactions and cannot maintain ledgers whose entropy (information content) is more than S. Thus replication poses a fundamental restriction on such blockchain designs. In practice, the computational burden increases with the number of nodes (example: mining puzzles get harder as time progresses and more users participate) and a more nuanced model capturing this phenomenon would imply that λ is more tightly upper bounded as n increases.

On the other hand, the information theoretic bounds offer a drastically different view. These bounds are not specific to a method or technology. This independence lends them a fundamental

value and are derived as follows. Suppose all computation and storage, currently decentralized across the n nodes, can be used in a centralized manner. This allows a total of nT computation and nS storage. Thus the following upper bounds holds for any decentralized technology:

$$\text{Information theoretic upper bounds for blockchains :} \qquad \lambda \leq nT; \qquad \gamma \leq nS,$$

Thus there is a huge gap between the traditional designs that operate based on replication and the potential scalability of blockchain systems. Addressing this gap and looking to close it via innovative designs is the goal of this chapter. The key idea we introduce is the notion of a *coded storage* and *coded computations* in the context of blockchains.

It is illustrating to first begin by discussing a potential scaling solution being discussed presently in the blockchain literature: *sharding*. The key idea is to break up (or "shard") the blockchain into fragments which are then replicated; storage requirements are naturally reduced and computation required for validating transactions is also reduced by a factor equal to the number of shards k. However, the tradeoff is that the adversarial protection is also reduced by the same factor: if there are n nodes in total, then it is sufficient for an adversary to take over $\frac{n}{k}$ nodes to fully control a shard. Thus, in sharding, there is a significant tradeoff between the efficiency gain and adversarial protection capability. The fundamental limitations of this engineering solution sets the stage for innovative design of a fully-scalable blockchain technology.

The main contributions of this chapter are the following.

- Drawing upon principles from information and coding theory, we propose a radically different scalable design methodology called *coded sharding*.
- We propose a particular instantiation of the general coded-sharding principle called `PolyShard`, which stands for Langrange polynomial based coded-sharding.
- We prove that `PolyShard` provides both scalability and adversarial protection capability, ie., it is possible to get a factor n increase in efficiency (throughput / storage), while *simultaneously* maintaining the adversarial protection also as $\Theta(n)$.

5.1 Sharding: benefits and limitations

Sharding has been proposed as a methodology for scaling the computation and storage abilities of a blockchain payment system. The basic idea of sharding is to allocate users (who are using the payment system) into groups called shards, and each shard maintains a ledger of transactions between users inside the shard. Thus if there are k shards, then there are k independent ledgers, each comprising of transactions between distinct users. Similarly, each processing node in the blockchain payment system is allocated into a shard, thus if there are n processing nodes, then there are $\frac{n}{k}$ nodes allocated to each shard on an average.

The main advantage of sharding is that the computation and storage of the blockchain payment system grows linearly in the number of shards k, since in effect, there are k independent blockchains. We can also parameterize level of sharding by $f = \frac{n}{k}$, the number of nodes participating in a single shard. Given that the blockchain has essentially been split into k parts, throughput and the entropy of the blockchain payment system grow linearly in the number of shards, i.e.,

$$\text{Sharding performance for blockchains :} \qquad \lambda = kT = \frac{nT}{f}; \qquad \gamma = kS = \frac{nS}{f}.$$

There are two main issues related to sharding:

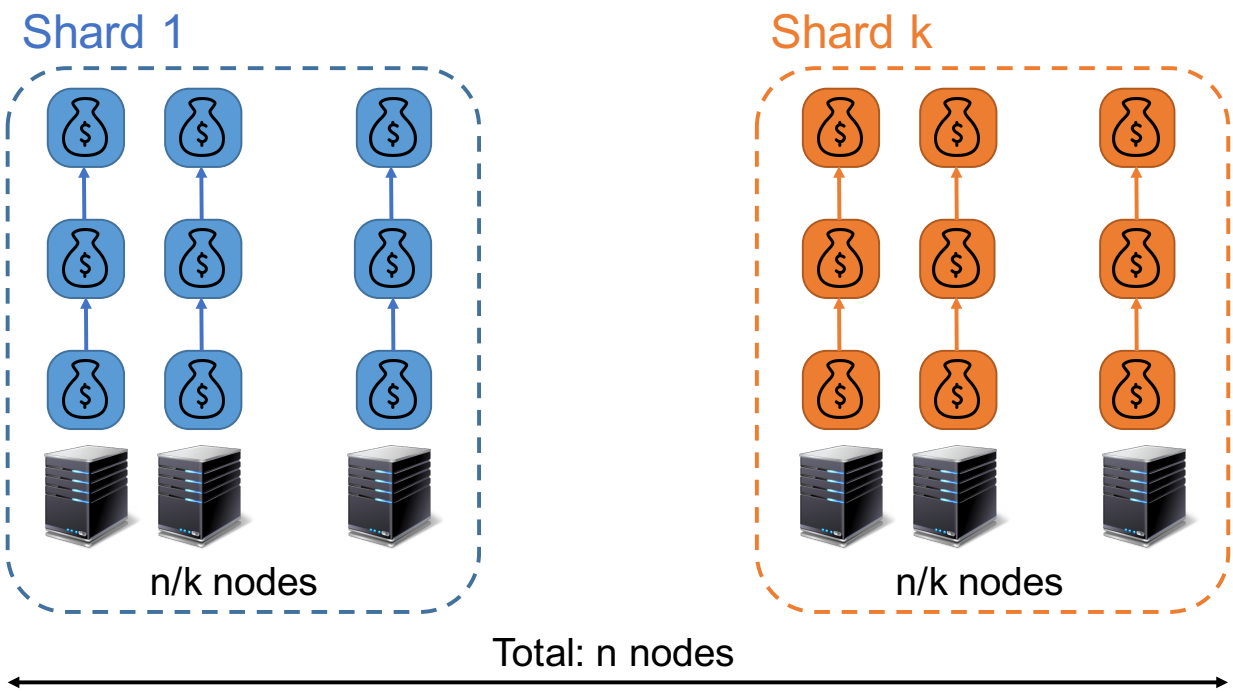


Figure 5.1: Sharding

- *Cross-shard transactions* need to be handled separately, since existing nodes have been assigned to specific shards. The resulting engineering solution adds complexity to the overall scheme and lowers the performance (throughput and entropy).
- The more the number of shards (i.e., the larger the value of k), the more parallelism is achieved and performance (both throughput and entropy) improves. However, this lowers the decentralization being achieved and corresponding trust is lowered. The fundamental *trust-performance tradeoff* needs to be handled as a result of allocating processing nodes to shards.

5.1.0.1 Cross-shard transactions

When users are allocated randomly into shards, the probability that a transaction between two random users is a cross-shard transaction is $\frac{k-1}{k}$, which approaches 1 when k is large. Thus most of the transactions happen across shards and a mechanism needs to be in place in order to deal with cross-shard transactions. The current proposals for cross-shard transactions [118, 106] implement a locking mechanism where the money is locked up from the source-shard in an escrow till it is released into the destination-shard. Designing lock-free mechanisms for cross-shard transactions is a major engineering direction of ongoing research. We note that any solution will still not alleviate the loss of trust that arises when k gets larger; this is discussed next.

5.1.0.2 Trust-scaling tradeoff

In order to achieve linear scaling using sharding, the number of nodes f participating in a shard has to be held constant. When this is the case, an adversary wishing to take control of a shard only needs to control f nodes, which are running validations and maintaining the ledger for a given shard

in order to compromise that shard. The ratio of adversarial nodes that can be tolerated becomes exceedingly small, i.e., the adversary should control strictly less than f nodes or strictly less than $\frac{f}{n}$ fraction of nodes, which approaches zero as n becomes large. Thus, *sharding achieves linear-scaling of throughput at the cost of decreasing the security guarantees.*

We note that existing sharding schemes propose some methods to alleviate the security issue: for example, by choosing random nodes through a cryptographic primitive to participate in a given shard, and using methods of shard rotation [106]. However, a powerful adversary can easily take control of a given shard by corrupting the users *after they have been chosen to participate in the shard*. Moreover, when the nodes are not assumed to be honest, but only rational, it might be in the interest of the small number of nodes in a given shard to *collude* to get illegal gains. Thus, sharding in its present form trades off trust for throughput, i.e., the larger the f, the more the security whereas lesser the scaling of throughput and computational abilities of the block-chain payment system (which scale like $k = \frac{n}{f}$). An important question is whether such a trust-scaling tradeoff is fundamental to decentralized blockchain payment systems or whether clever methodologies can achieve different scaling behaviors.

We note that in distributed systems theory, Byzantine consensus [110, 42] is a fundamental problem in distributed computing and cryptography. It has been used to build fault tolerant systems such as distributed storage systems [151, 41, 16, 108, 13, 49, 90], certificate authorities [147, 191], fair peer-to-peer sharing [181], and more recently cryptocurrencies [105, 124, 139, 15]. It has also been frequently used as building blocks in cryptographic protocols such as secure multi-party computation [86, 26].

In all these applications, for the most part, byzantine consensus has been handled using replication. For example, replicated state machine is a formalism [157, 109], where the intention is to run a state machine on a group of nodes, some of which are strategic. The state machine is replicated in parallel on many nodes, which attain byzantine consensus among themselves on the next instruction to be run. Thus replicated state machine is a formalism of which a blockchain system can be treated as a special case. Even in this formalism, the same issue as in blockchain exists, i.e., the computational and storage performance does not scale with the number of nodes participating but is rather constrained to be at most a constant, since the system is run based on full replication. The question of whether scaling and trust have a fundamental tradeoff is open in this setting too. We begin this long research program by addressing the fundamental limits of trust and performance in a simple decentralized setting next.

The main questions that arise in the context of scaling blockchain can be summarized as follows.

1. Is it possible to achieve a performance scaling linearly in T and S ?
2. How to best navigate the trust-throughput tradeoff?
3. Can we build "coded" state-machines as a general formalism instead of replicated state machines?

5.2 Coding vs. Replication

The general question of whether systems that can achieve better performance than replication have been well studied in information and coding theory. We begin with a simple problem of storage, weeding away many details of blockchain systems. Consider a storage system where there are k files $X_1, .., X_k$ of equal size that need to be stored in n nodes, where each node can store 1 file. The trivial solution for the problem when $n = fk$, with $f \geq 1$ is to store each file in f nodes; we call such a system as a replicated system.

We consider a simple erasure failure model where some nodes get erased. There are two formalisms for such an erasure model: random erasures and adversarial erasures. In the random erasure model, each storage node can get erased randomly with probability p. In the adversarial erasure model, at most p fraction of nodes can be erased by an adversary, who can handpick the nodes to be erased given other details.

In the random erasure model when using replication, the probability that all replicas of a given file get erased is p^f. If we require the probability of this failure mode to go to zero, we need $f \to \infty$. The total number of files stored in the system is $\frac{n}{n}$. Thus there is a tradeoff between the failure probability of the system, which is upper bounded by kp^f and the total storage of the system, which is $\frac{n}{f}$. If the failure-probability is required to go to zero, then f has to grow to infinity, and the total storage in the system does not scale linearly.

Shannon, in his celebrated work, showed that such a tradeoff *is not* fundamental and can be circumvented by the utilization of carefully crafted codes when n is large [162]. In a coded system, the total storage can scale as $\frac{n}{f}$, with a fixed constant f, even with probability of failure going to zero: in fact, according to Shannon's capacity theorem, $f = \frac{1}{1-p}$ is the sharpest such constant. Thus codes can save significantly over replication in the reliability-storage tradeoff. This insight has led to the penetration of coding into various aspects of modern information storage and retrieval when the failure mode is randomized.

In the adversarial erasure model, a fraction p of nodes are erased by the adversary, i.e., pn nodes are erased. This necessitates each file to be replicated in more than pn nodes. Thus the maximum number of files that can be stored with guaranteed recovery is $\frac{1}{p}$ irrespective of the number of nodes n. However, if the file size is large, then maximum-distance-separable codes such as Reed-Solomon codes ensure that in n nodes, $k = (1-p)n$ files can be stored with guaranteed recovery against an adversary. Thus coding can have a factor n gain over replication in storage ability under adversarial failure mode. This highlights the great benefits that coded systems can bring in blockchain payment systems, where strongly adversarial users can exist.

An important question is what role such coding methodologies can play in blockchains. We envision two roles for coding in blockchain.

1. Codes can be used to store information pertaining to the blockchain in such a way that it can tolerate an optimal number of adversaries as well as errors and erasures.

2. It may be possible for transaction validation to be coded so that computation can be performed in the coded domain as well.

5.3 Coded Sharding

We now present the concept of coding for sharding in blockchains (i.e., coded sharding). In particular, we propose `PolyShard`: "polynomially coded sharding" scheme that achieves information-theoretic upper bounds on throughput and storage of blockchains (i.e., λ and γ) as well as on trust, thus enabling a truly scalable system. More precisely, in a system composed of n nodes, we show that `PolyShard` provides both scalability and adversarial protection capability, i.e., it is possible to get a factor n increase in efficiency (throughput / storage), while *simultaneously* maintaining the adversarial protection also as $\Theta(n)$.

`PolyShard` is rooted in recent developments in *coded computing*, in particular Lagrange Coded Computing [187], which provide a transformative framework for injecting computation redundancy in unorthodox coded forms in order to deal with failures and errors in distributed computing. The key idea of LCC is to encode data using the well-known Lagrange interpolation polynomial, in order

to create computation redundancy to provide security against malicious servers. We will leverage LCC to develop our coded sharding technique. We start by formalizing the system model that we consider for sharding.

5.3.1 System Model

We consider a blockchain system partitioned into k indepedent shards, each of which maintains a sub-chain that records a part of the past transactions. Each client of the system is associated to a particular shard of the chain. For clarity of presentation we focus on transactions that are verifiable intra-shard; cross-shard verifications are an added complexity complementary to our contributions here; for instance, the atomic payment and locking mechanisms of [106] can be naturally incorporated with the proposed ideas. We define the computation and networking models below.

5.3.1.1 Computation model

Each shard j, $j = 1, 2, \ldots, k$, maintains its own sub-chain. We use t to denote discrete time in rounds and $Y_j(t) \in \mathbb{U}$ denotes the verified block in round t at shard j, where $\mathbb{U}$ is a vector space over a field $\mathbb{F}$. Then the sub-chain at shard j till time $t-1$ is denoted as $Y_j^{t-1} = (Y_j(1), Y_j(2), \ldots, Y_j(t-1))$. We wish to validate a new block $X_j(t) \in \mathbb{U}$ proposed in shard j in round t with respect to the past of the sub-chain Y_j^{t-1} to ensure that it does not contain any double-spends or other irregularities. We abstract out the mechanism which generates the proposal and instead focus on validating the proposed transaction. We note that the proposal can be generated by a small number of nodes, while security depends on the number of nodes that validates the transactions. Note that since transactions are intra-shard, the sub-chain is sufficient to verify whether $X_j(t)$ is valid. To verify $X_j(t)$, shard j computes a verification function $f^t : \mathbb{U}^t \to \mathbb{V}$, over $X_j(t)$ and the sub-chain Y_j^{t-1}, for some vector space $\mathbb{V}$ over $\mathbb{F}$.

Having obtained $h_j^t = f^t(X_j(t), Y_j^{t-1})$, shard j computes an indicator variable z_j^t, such that $z_j^t = \mathbb{1}(h_j^t \in \mathcal{W})$, where $\mathcal{W} \subseteq \mathbb{V}$ denotes the subset of the values of the verification function for which the block is valid. A simple example is when $\mathbb{V} = \{0, 1\}$, the output is binary and denotes the validity, i.e., $\mathcal{W} = \{1\}$. Finally, the verified block $Y_j(t)$ is computed as $Y_j(t) = z_j^t X_j(t)$, i.e., if the block is not valid, it is treated as a null block which is represented as 0 in the field.

Without loss of generality, we model the verification function f^t as a multivariate polynomial of degree d, motivated by the following result [193]: Any Boolean function $\{0, 1\}^n \to \{0, 1\}$ can be represented by a polynomial of degree $\leq n$ with at most 2^{n-1} terms. Due to this result, common verification functions, such as balance check and digital signature verification, can all be transformed into polynomials.

Example: Balance verification. We consider a simple payment blockchain system that keeps records of the balance transfers between clients. We assume that there are M clients in each shard, for some constant M that does not scale with t. In this scenario, a block submitted to shard k at time t, $X_j(t)$, consists of multiple transactions, and is represented by a pair of real vectors $X_j(t) = (X_j^{\text{send}}(t), X_j^{\text{receive}}(t))$, for some $X_j^{\text{send}}(t), X_j^{\text{receive}}(t) \in \mathbb{R}^M$. A transaction that reads "Client p sends s dollars to client q." will deduct s from $X_j^{\text{send}}(t)[p]$ and add s to $X_j^{\text{receive}}(t)[q]$. Clients that do no send/receive money will have their entries in the send/receive vectors set to zeros. To verify $X_j(t)$, we need to check that all the senders in $X_j(t)$ have accumulated enough unspent funds from previous transactions. This naturally leads to the following verification function.

$$f^t(X_j(t), Y_j^{t-1}) = X_j^{\text{send}}(t) + \sum_{i=1}^{t-1} (Y_j^{\text{send}}(i) + Y_j^{\text{receive}}(i)). \tag{5.1}$$

Here the verification polynomial f^t has constant degree $d = 1$, with $\mathbb{F} = \mathbb{R}$, $\mathbb{U} = \mathbb{R}^M$, and $\mathbb{V} = \mathbb{R}^M$. We claim the block $X_j(t)$ valid if none of the entries of the above verification function is negative, and we set $Y_j(t) = X_j(t)$ and append it the sub-chain of shard k. Otherwise, we append a dummy all-zero block.

5.3.1.2 Networking model

The above blockchain system is implemented distributedly over n network nodes. A subset $\mathcal{M} \subset \{1, 2, \ldots, n\}$ of these nodes may be malicious/adversarial, and the malicious nodes compute and communicate arbitrarily erroneous results during the process of block verification. Only the honest nodes will follow the designed networking protocol described below.

At time t, each node i, $i = 1, 2, \ldots, n$, locally stores some data, denoted by $Z_i^{t-1} = (Z_i(1), Z_i(2), \ldots, Z_i(t-1))$, where $Z_i(j) \in \mathbb{W}$ for some vector space $\mathbb{W}$ over $\mathbb{F}$. The locally stored data Z_i^{t-1} is generated from all shards of the blockchain using some function ϕ_i^{t-1}, i.e.,

$$Z_i^{t-1} = \phi_i^{t-1}(Y_1^{t-1}, Y_2^{t-1}, \ldots, Y_k^{t-1}). \tag{5.2}$$

Next, given the k incoming blocks $\{X_j(t)\}_{j=1}^k$, each node i computes an intermediate result g_i^t using some function ρ_i^t on the incoming blocks and its local storage, such that,

$$g_i^t = \rho_i^t(X_1(t), X_2(t), \ldots, X_k(t), Z_i^{t-1}), \tag{5.3}$$

and then broadcasts the result g_i^t to all other nodes.

Having received all the broadcast messages, each node i decodes the verification results for all k shards $\hat{h}_{1i}^t, \hat{h}_{2i}^t, \ldots, \hat{h}_{ki}^t$, using some function ψ_i^t, i.e.,

$$(\hat{h}_{1i}^t, \hat{h}_{2i}^t, \ldots, \hat{h}_{ki}^t) = \psi_i^t(g_1^t, g_2^t, \ldots, g_n^t). \tag{5.4}$$

Using these decoded results, node i computes the indicator variables $(\hat{z}_{1i}, \hat{z}_{2i}, \ldots, \hat{z}_{ki})$, and then the verified blocks $\hat{Y}_{ji}(t) = \hat{z}_{ji}^t X_j(t)$, for all $j = 1, 2, \ldots, k$.

Finally, each node i utilizes the verified blocks to update its local storage using some function χ_i^t, i.e., $Z_i^t = \chi_i^t(\hat{Y}_{1i}(t), \hat{Y}_{2i}(t), \ldots, \hat{Y}_{ki}(t), Z_i^{t-1})$.

5.4 PolyShard

We now describe `PolyShard`, our proposed coded sharding scheme, which utilizes coding in *both* storage and transaction verification as described below.

5.4.1 Storage encoding in `PolyShard`

The first key property of `PolyShard` is that each node stores a *coded* version of the sub-chains (as opposed to naive uncoded replication approaches). The encoding of sub-chain history is done via utilization of Lagrange polynomials (hence the name `PolyShard`). More specifically, we pick k distinct elements $\omega_1, \omega_2, \ldots, \omega_k \in \mathbb{F}$, and create the following Lagrange polynomial

$$u^{t-1}(z) = \sum_{j=1}^{k} Y_j^{t-1} \prod_{\ell \neq j} \frac{z - \omega_\ell}{\omega_j - \omega_\ell}. \tag{5.5}$$

Note that $u^{t-1}(\omega_j) = Y_j^{t-1}$ for all $j = 1, 2, \ldots, k$.

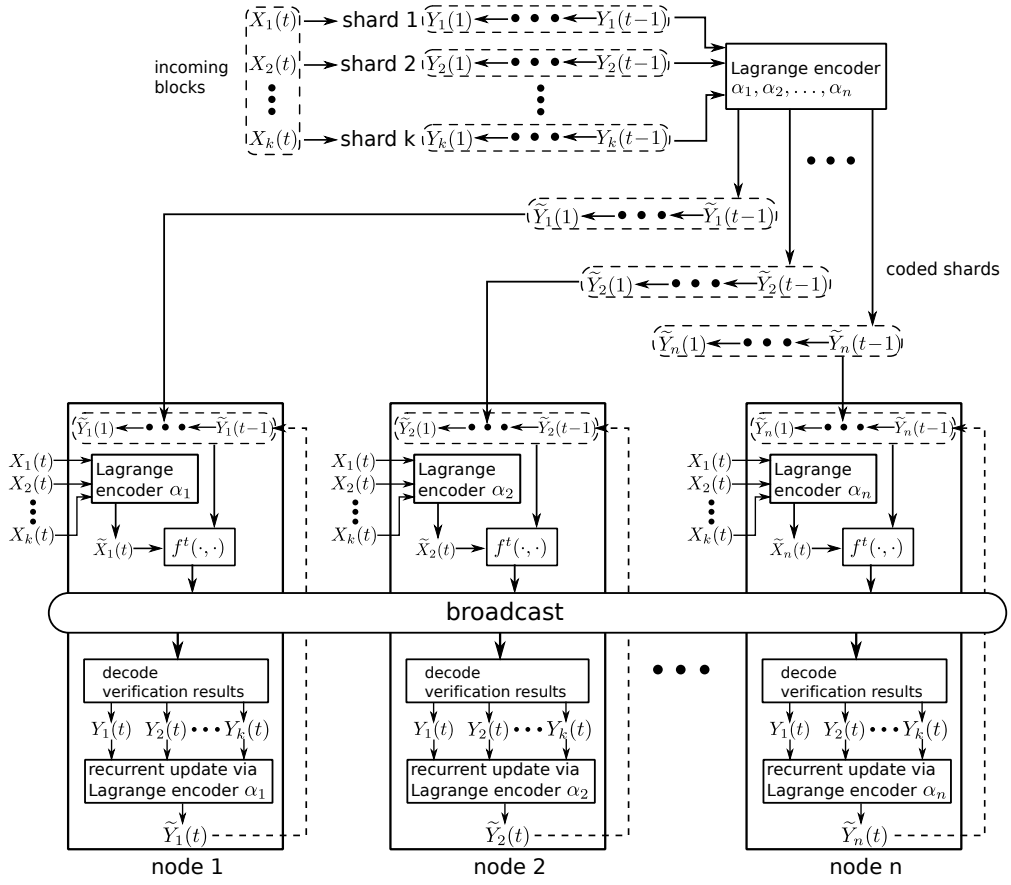


Figure 5.2: Illustration of PolyShard scheme.

Next, as shown in Figure 5.2, we pick n distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbb{F}$, one for each node. This creates n coded sub-chains, denoted by $\tilde{Y}_1^{t-1}, \tilde{Y}_2^{t-1}, \ldots, \tilde{Y}_n^{t-1}$, by evaluating the above $u^{t-1}(z)$ at the points $\alpha_1, \alpha_2, \ldots, \alpha_N$. That is, for all $i = 1, 2, \ldots, n$,

$$\tilde{Y}_i^{t-1} = u^{t-1}(\alpha_i) = \sum_{j=1}^{k} Y_j^{t-1} \prod_{\ell \neq j} \frac{\alpha_i - \omega_\ell}{\omega_j - \omega_\ell} = \sum_{j=1}^{k} c_{ij} Y_j^{t-1}, \tag{5.6}$$

for all $i = 1, 2, \ldots, n$. We note that $\tilde{Y}_i^{t-1}$ is encoded as a linear combination of the uncoded sub-chains $Y_1^{t-1}, Y_2^{t-1}, \ldots, Y_k^{t-1}$, and the coefficients c_{ij}s do not depend on the time index t. Having generated these coded sub-chains, we store $\tilde{Y}_i^{t-1}$ onto node i. For this encoding to be viable, we need large enough field such that $|\mathbb{F}| \geq n$. For small finite fields (e.g., binary field), we can enable this encoding scheme by embedding each data element onto an extension field $\mathbb{F}^m$ with $|\mathbb{F}|^m \geq n$, and then apply the coding in $\mathbb{F}^m$.

An important advantage of storage encoding in `PolyShard` is that, upon addition of newly verified transactions in sub-chains, the encoded storage at each node needs to be updated *incrementally* without the need to re-encode the entire history.

More specifically, upon the addition of new verified transactions $Y_j(t)$, $j = 1, 2, \ldots k$, we first create

$$\tilde{Y}_i(t) = \sum_{j=1}^{k} c_{ij} Y_j(t), \tag{5.7}$$

and append $\tilde{Y}_i(t)$ into the local coded sub-chain of node i (i.e., $\tilde{Y}_i^{t-1}$) to update it to

$$\tilde{Y}_i^{t} = (\tilde{Y}_i^{t-1}, \tilde{Y}_i(t)). \tag{5.8}$$

Note that, since the set of coefficients c_{ij}s in (5.7) are identical to those in (5.6), appending a coded block to a coded sub-chain is equivalent to appending uncoded blocks to the uncoded sub-chains, and then encoding from the updated sub-chains. This commutativity between sub-chain growth and storage encoding allows each node to update its local sub-chain by only utilizing the newly verified blocks.

Another important advantage of the `PolyShard` storage encoding is that it is *oblivious* of the verification function, i.e., the coefficients c_{ij} are independent of f^t. Therefore, the data encoding can be carried out independently of the verification, and the same coded storage can be simultaneously used for all different types of verification items, which could include verifying account balances, digital signatures or checking smart contracts.

5.4.2 Coded verification in `PolyShard`

At time t, k blocks $X_1(t), X_2(t), \ldots, X_k(t)$ are submitted to the k shards for verification over n network nodes. Note that in `PolyShard` each node stores a coded version of the sub-chains history, hence all computations that nodes will do to verify transactions will be over coded sub-chains, which makes the verification challenging.

The `PolyShard` scheme verifies these blocks in three steps as described below (also illustrated in Figure 5.2).

Step 1: block encoding. Having received the k blocks, each node i, $i = 1, 2, \ldots, n$, computes a coded block $\tilde{X}_i(t)$ as a linear combination of the blocks using the same set of coefficients as in (5.6).

That is,

$$\tilde{X}_i(t) = \sum_{j=1}^{k} c_{ij} X_j(t). \tag{5.9}$$

We note that this block encoding operation at node i can be also viewed as evaluating the polynomial $u_t(z) = \sum_{j=1}^{k} X_j(t) \prod_{\ell \neq j} \frac{z-\omega_\ell}{\omega_j - \omega_\ell}$ at the point α_i.

Step 2: local *coded* computation. Each node i applies the function f^t to the *coded* block $\tilde{X}_i(t)$, and its locally stored coded sub-chain $\tilde{Y}_i^{t-1}$ to compute

$$g_i^t = f^t(\tilde{X}_i(t), \tilde{Y}_i^{t-1}). \tag{5.10}$$

Having finished the local computations, each node ℓ broadcasts its computation result g_i^t to all other nodes.

Step 3: decoding. Using the computation results $g_1^t, g_2^t, \ldots, g_n^t$, a maximum β fraction of which may be erroneous from malicious nodes, each node aims to decode the intended results $\{f^t(X_j(t), Y_j^{t-1})\}_{j=1}^k$. We note the following key points about steps 1 and 2.

1. The computation result g_i^t at node i can be viewed as the evaluation of $f^t(u_t(z), u^{t-1}(z))$ at the point α_i.
2. $f^t(u_t(z), u^{t-1}(z))$ is a univariate polynomial of degree $(k-1)d$.
3. The intended results $\{f^t(X_j(t), Y_j^{t-1})\}_{j=1}^k$ can be recovered by evaluating the polynomial $f^t(u_t(z), u^{t-1}(z))$ at $\{\beta_j\}_{j=1}^k$.

From the above three points, we conclude that the decoding problem can be viewed as "recovering a univariate polynomial of degree $(k-1)d$ from its evaluations at n distinct points with up to β fraction of being erroneous".

By mapping the verification problem to polynomial interpolation, the above decoding problem can now be solved optimally using the process of decoding a Reed-Solomon code with dimension $(k-1)d+1$ and length n (see, e.g., [154]). In order for this decoding to be successful, the number of malicious nodes βn cannot exceed $(n-(k-1)d)/2$. In other words, the genuine network nodes can successfully decode $f^t(u_t(z), u^{t-1}(z))$ only if the number of shards k is upper bounded as

$$k \leq \frac{(1-2\beta)n}{d}. \tag{5.11}$$

5.4.3 Optimality of PolyShard

As we discussed in Section 5.1, state-of-the-art uncoded sharding schemes cannot achieve linear throughput scaling, while guaranteeing security against a constant fraction β of adversary nodes. In fact, if we want to maintain security against β fraction of nodes, then the number of shards can at most be $k = \frac{n}{\beta n} = \frac{1}{\beta}$ (since there should be at least βn nodes in each shard. Hence, we can only achieve throughput of

$$\lambda_{\text{uncoded sharding}} = kT = \frac{T}{\beta}, \tag{5.12}$$

which is a constant.

Our main result is that PolyShard can achieve linear throughput scaling while still guaranteeing security against a constant fraction of the nodes, without increasing the storage and computation overhead of the system. Our main result is summarized in the following theorem.

Theorem 1. *For a blockchain system consisting of multiple shards, each of length t, a polynomial verification function f^t with constant degree d, operated on n network nodes, up to β (for some constant $0 \leq \beta < \frac{1}{2}$) fraction of which may be malicious, the above described* `PolyShard` *scheme achieves the throughput of*

$$\lambda_{\texttt{PolyShard}} = \frac{(1-2\beta)Tn}{d} = \Theta(n), \tag{5.13}$$

and total storage of

$$\gamma_{\texttt{PolyShard}} = \frac{(1-2\beta)Sn}{d} = \Theta(n), \tag{5.14}$$

while guaranteeing security against β fraction of the nodes.

Comparing the achievable throughput and storage of `PolyShard` (i.e., equations (5.13) and (5.14)) with the information-theoretic upper bounds for blockchains described at the beginning of this chapter, we notice that `PolyShard` is order-wise optimal *simultaneously* in security, storage, and throughput. Compared with the state-of-the-art uncoded sharding schemes that only achieve a constant throughput (see eq. (5.12)), we also note that `PolyShard` provides a storage and throughput gain that scales linearly with the number of nodes n.

5.5 Simulation Results

We perform detailed simulations to assess the performance of `Polyshard` in the payment blockchain system described as an example. This system keeps records of all the balance transfers between clients, and verifies new blocks by comparing them with the sum of the previously verified blocks (i.e., computing the verification function in (5.1)). More specifically, the system contains k shards, each managing M clients. At each time epoch t, one block of transactions is submitted to every shard j. We simulate this system over n nodes using the full replication, uncoded sharding, and `PolyShard` schemes respectively.

We measure the throughput of each scheme under different values of n and t to understand its scalability. Throughput is defined as the number of blocks verified per time unit, and is measured by dividing k (the number of blocks generated per epoch) by the average verification time (to be measured) of the n nodes. For `PolyShard`, the verification time also includes the time each node spent on encoding the blocks. However, since the encoding time is a constant, whilst the balance summation time increases with t as the chain becomes longer, it is expected that the encoding time is becoming negligible.

We note that the storage efficiency and security level of each scheme are decided by system parameters and, thus, do not need measurements.

We simulate this system for $t = 1000$ epochs, using different number of shards $k \in [5, 50]$. Each shard manages $M = 2000$ clients. We fix the ratio $n/k = 3$. Thus, the number of nodes is $n \in [15, 150]$. We plot the relation between t and throughput when $n = 150$ in Figure 5.3, and the relation between n and throughput when $t = 1000$ in Figure 5.4.

Results and discussions

1. Throughput: As expected, `PolyShard` provides the same throughput as uncoded sharding, which is about K times of the throughput of full replication. From Figure 5.3, we observe that the throughput of all three schemes drops as the time progresses. This is because that the computational complexity of verifying a block increases as more blocks are appended to each shard. In terms of scalability, Figure 5.4 indicates that the throughput of `PolyShard`

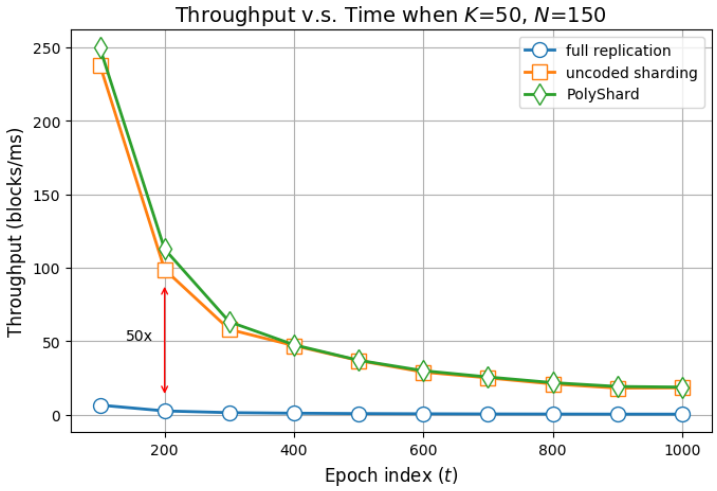


Figure 5.3: Throughput of the three verification schemes; here number of nodes n =150.

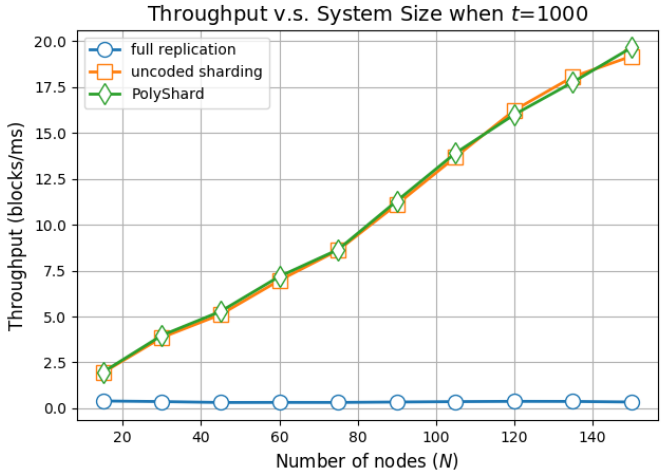


Figure 5.4: Throughput of the three verification schemes; here number of epochs t =1000.

and uncoded sharding both increases linearly with the network size n (and k), whilst the throughput of full replication almost stays the same.

2. Storage: It is straightforward to see that `PolyShard` provides the same storage gain over full replication as uncoded sharding, with a factor of k. Thus, `PolyShard` and uncoded sharding are scalable in storage, but full replication is not (Table 5.1).

3. Security (# of malicious nodes that can be tolarated): Full replication can tolerate up to 50% of malicious nodes, achieving the maximum security level $\beta_{\text{full}} = \frac{n}{2}$. The error-correcting process of `PolyShard` provides robustness to $\beta_{\texttt{PolyShard}} = \frac{n-k}{2} = \frac{n-n/3}{2} = \frac{n}{3}$ malicious nodes. In contrast, under uncoded sharding, each shard is only managed by 3 nodes. Thus, its security level is only 1 regardless n, which is not scalable (Table 5.2).

N	15	30	60	90	120	150
γ_{full}	1	1	1	1	1	1
γ_{sharding}	5	10	20	30	40	50
$\gamma_{\texttt{PolyShard}}$	5	10	20	30	40	50

Table 5.1: Storage of the three schemes under different network size n.

N	15	30	60	90	120	150
β_{full}	7	15	30	45	60	75
β_{sharding}	1	1	1	1	1	1
$\beta_{\texttt{PolyShard}}$	5	10	20	30	40	50

Table 5.2: Security of the three schemes under different network size n.

In summary, `PolyShard` outperforms both full replication and uncoded sharding because it is the only scheme that can simultaneously 1) alleviate the storage load at each node; and 2) boost the verification throughput by scaling out the system, and 3) without sacrificing the safety requirement even when the number of adversaries also grows with network size.

5.6 Discussion

In this section, we discuss how `PolyShard` fits into the overall architecture of a contemporary blockchain system.

5.6.1 Integration into blockchain systems

We note that `Polyshard` has so far been described in a simple setting where each shard produces one block in lock-step. We highlight one instantiation of how `Polyshard` could fit into the architecture of an existing blockchain system, which combines a standard sharding method for proposal followed by `Polyshard` for finalization. The K shards are obtained by assigning users to shards via a user-assignment algorithm. The N nodes are partitioned into K shards using a standard sharding system (see [106]). Inside of the shard, the nodes run a standard blockchain along with a finalization algorithm to get a *locally finalized* version of the block.

Each node is also assigned a coded shard via a coded-shard-assignment algorithm, which assigns a random field element $\alpha_i \in \mathbb{F}$ to a node so that the node can compute which linear combination it will use for coding. We point out here that it is easy to handle churn (users joining and leaving) by this method if the size of the finite field $\mathbb{F}$ is much larger than N - since at this point, the probability of collision (two users getting assigned the same field element) becomes negligible. Thus each node plays a role in both an uncoded shard as well as a coded shard, thus its storage requirement will be doubled; however, our system still has storage efficiency scaling with N. The `Polyshard` algorithm now gets the locally finalized blocks from the different shards at regular intervals and it acts as a global finalization step performing coded validation at the level of the locally finalized blocks. We point out that users requiring high trust should wait for this global finalization stamp before confirming a payment, whereas users requiring short latency can immediately utilize the local-finalization for confirmation.

Beyond the aforementioned issues, there may be cross-shard transactions present in the system, which are payments or smart contracts with inputs and outputs distributed across multiple shards. In such a case, we will use a locking-based method, which locks the payment at the source shard and produces a certificate to the destination shard so that the amount can be spent; this idea has been proposed as well as implemented in `Elastico` [118] and `Omniledger` [106].

5.6.2 Modelling cross-shard transactions

So far we have focused on verifying blocks within individual shards. One way to model the cross-shard transactions is that for each incoming block, we compute the verification function over all shards. Specifically, for an incoming block $X_k(t)$ submitted to shard k in time t, we generate a particular verification function f_k^t for $X_k(t)$. Instead of being an argument of the verification function, $X_k(t)$ has now become one of the components that define f_k^t. To verify $X_k(t)$ across shards, we need to compute $(f_k^t(Y_1^{t-1}), f_k^t(Y_2^{t-1}), \ldots, f_k^t(Y_K^{t-1}))$, and use all these results to decide if $X_k(t)$ is valid or not. The proposed `PolyShard` scheme can be easily adapted to be used in this model.

5.6.3 Relationship to verifiable computing

An alternative paradigm for accelerating computing in blockchain is verifiable computing [80, 33, 135, 30, 27], where a single node executes a set of computations (for example, payment validation) and integrity of these computations are then cryptographically certified. A major difference between our framework and verifiable computing is that our scheme is *information-theoretically secure* against a computationally unbounded adversary as against the computational security offered by verifiable-computing schemes. However, verifiable computing schemes can provide zero-knowledge proofs, whereas our scheme does not offer zero-knowledge capability. Finally, verifiable computing is relevant in an asymmetric setting, where one computer is much more powerful than the others, unlike `Polyshard` which is designed for a symmetric setup comprising of equally powerful and decentralized nodes.

5.6.4 Future research directions

Polyshard currently works with polynomials whose degree scales sub-linearly with the number of nodes. An interesting direction of future work is to remove this limitation. In particular, computations that can be represented as *low-depth* arithmetic circuits can be implemented iteratively using low-degree polynomials. Another important direction of future research is the design of validation schemes that can be represented as low-degree polynomials or low-depth arithmetic circuits.

Since the encoding and decoding processes involve polynomial evaluation and interpolation, the main advantage of verifiable computing schemes is to enable utilizing efficient polynomial algorithms, by performing all the coding operations at a single node. For example, for a single node to generate all coded blocks $\tilde{X}_1(t), \tilde{X}_2(t), \ldots, \tilde{X}_N(t)$, it can use fast multi-point polynomial evaluation algorithms to evaluate $u_t(z)$ at the points $\alpha_1, \alpha_2, \ldots, \alpha_N$ with a computational complexity $O(N \log^2 N \log\log N)$ (see, e.g., [17, 180]), instead of $O(NK)$ using `PolyShard`. An interesting future research direction is to develop verifiable computing schemes for `PolyShard` that achieve a total coding overhead that scales (almost) linearly with the network size N.

5.7 Acknowledgement

The authors thank S. Li, and M. Yu for collaborations on the design of `Polyshard`. This chapter includes and expands on material in [115].

Chapter 6

Spider: Efficient Routing for Payment Channel Networks

Mohammad Alizadeh, MIT
Giulia Fanti, CMU
Pramod Viswanath, UIUC

Today's cryptocurrencies have poor transaction throughput and slow confirmation times. Bitcoin supports 3-7 transactions per second and takes tens of minutes to confirm a transaction [178]. By comparison, established payment systems like Visa process thousands of transactions per second with a delay of a few seconds [178]. Further, high transaction costs make current blockchains impractical for micropayments. The median Bitcoin transaction fee regularly exceeds $1 and reached $34 in December 2017 [2].

Payment channel networks are a promising solution to these scalability challenges. A payment channel is a blockchain transaction that escrows a given user Alice's money in order to enable future transactions to a specific recipient (Bob), much like a gift card. Once Alice opens a payment channel to Bob, she can transfer funds repeatedly and securely without recording every transaction on the blockchain. After some time, Alice or Bob can close the payment channel and record only the final state of the channel, rather than the full sequence of transactions. A payment channel network consists of bidirectional payment channels between pairs of participants (e.g., end-users, merchants). By routing payments through intermediate payment channels, two users can seamlessly transfer funds even if they do not share a direct payment channel. First proposed in the Lightning Network [143], payment channel networks have attracted considerable interest in the cryptocurrency community [91, 173, 121]. Multiple implementations are under development (e.g., Bitcoin's Lightning Network [143], Ethereum's Raiden Network [10]). In July 2018, a pilot program started to allow over 100 merchants to accept payments over the Lightning Network [51].

Payment channel networks route cryptocurrency, not data, but their design presents several technical and economic challenges familiar to communication networks. First, payment channel networks require efficient network protocols to find paths for payments and to deliver them with high throughput and low delay. Efficient networking is essential to the economic viability of payment channel networks, as it directly impacts the amount of capital that needs to be escrowed in payment channels to support a certain transaction throughput. Second, the network must provide the right economic incentives to both end-users, who desire low transaction fees, and service providers, who

wish to maximize their profits from routing payments. Third, the network should ensure the privacy of user transactions.

Although some of these design challenges are similar to communication networks, current networking solutions for payment channel networks are primitive compared to more mature networks like the Internet. For example, to make a payment in the Lightning Network, senders must first find a path with enough funds to *fully* satisfy the payment. This approach resembles "circuit switching" in early analog telephone networks, where, to make a call, telephone exchanges first had to establish a dedicated wire circuit between two telephones. Circuit switching has long been recognized to be inefficient in communication networks, as it prevents effective multiplexing of network bandwidth between different users and applications. Similarly, in payment channel networks, circuit switching makes it difficult to use payment channel funds effectively to transfer a mix of large and small payments. This is but one example where payment channel networks can benefit from lessons learned in the design of the Internet.

In this chapter, we revisit the design of payment channel networks in view of the decades of research that has led to the development of modern communication networks. We have two primary goals. First, we formalize central design problems such as routing and scheduling transactions in payment channel networks. Our analysis characterizes fundamental limits on key metrics, such as transaction throughput as a function of network parameters, such as the transaction pattern, network topology, amount of funds escrowed, etc., and provides a foundation for evaluating different network designs. Second, we use the insights from this analysis to propose *Spider*, a new network architecture designed from the ground up for payment channel networks.

Spider has several features that substantially improve its performance compared to existing payment channel networks. First, it uses packet switching: senders break up payments into *transaction units* and transmit them across different network paths over a period of time. Second, Spider uses a custom *transport* layer for payments that performs congestion control and reliable delivery functions analogous to the Transmission Control Protocol (TCP) [95, 46] protocol in the Internet. Third, Spider routers queue transaction units when they lack the funds to send them immediately, and use *scheduling* algorithms to decide in what order to service different transactions, similar to how routers in a data network share link bandwidth between different traffic flows when their rate exceeds link capacity. By combining congestion control and intelligent in-network payment scheduling, Spider achieves high utilization of payment channel funds while supporting a variety of classes of service for payment delivery (e.g., atomic and non-atomic payments with different deadlines). Lastly, Spider uses novel *imbalance-aware* routing algorithms that proactively rebalance payment channels. By incorporating imbalance mitigation explicitly into its routing, Spider minimizes the need for making expensive and slow on-chain transactions to rebalance payment channels.

6.1 Background

6.1.1 Payment Channels

Bidirectional payment channels are the building blocks of a payment channel network. A bidirectional payment channel allows a sender (Alice) to send funds to a receiver (Bob) and vice versa. To open a payment channel, Alice and Bob jointly create a transaction that escrows money for a fixed amount of time [143]. Suppose Alice puts three units in the channel, and Bob puts four (Fig. 6.1). Now, if Bob wants to transfer one token to Alice, he sends her a cryptographically-signed message asserting that he approves the new balance. This message is not committed to the blockchain; Alice simply holds on to it. Later, if Alice wants to send two tokens to Bob, she sends a signed message to Bob approving the new balance (bottom left, Fig. 6.1). This continues until one party decides to close

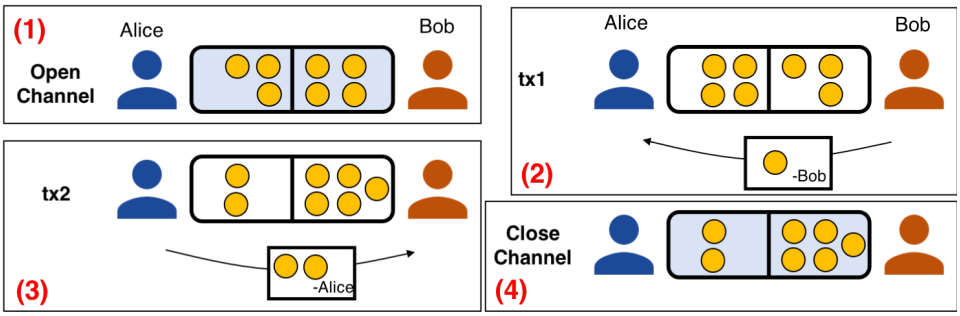


Figure 6.1: Bidirectional payment channel between Alice and Bob. A blue, shaded block indicates a transaction that is committed to the blockchain.

the channel, at which point they publish the latest message to the blockchain asserting the channel balance. If one party tries to cheat by publishing an earlier balance, the other party can override it with the signed message reflecting the later balance. Because intermediate transactions are not committed to the blockchain, Alice and Bob can transact many times without paying blockchain fees or waiting for long confirmation times.

6.1.2 Payment Channel Networks

A payment channel network is a collection of bidirectional payment channels (Fig. 6.2). If Alice wants to send three tokens to Bob, she first finds a path to Bob that can support three tokens of payment. Intermediate nodes on the path (Charlie) will relay payments to their destination. Hence in Fig. 6.2, two transactions occur: Alice to Charlie, and Charlie to Bob. To incentivize Charlie to participate, he receives a routing fee. To prevent him from stealing funds, a cryptographic hash lock ensures that all intermediate transactions are only valid after a transaction recipient knows a private key generated by Alice [143]. Once Alice is ready to pay, she gives that key to Bob; he can either broadcast it (if he decides to close the channel) or pass it to Charlie. In either case, Charlie learns the key, and he is incentivized to pass it upstream to Alice so that he can also get paid. To ensure that Charlie has enough time to claim his reward, transactions have expiration times that decrease at each hop (see [143] for details). When all participants cooperate, the payment occurs entirely off-chain and is very fast (limited only by network latency). The blockchain serves primarily as a settlement layer to close payment channels and to resolve disputes.

6.2 Related Work

The Lightning Network [143] was the first to show how cryptographic primitives like hashed timelock contracts enable the construction of secure payment channel networks for Bitcoin. Several projects have since developed payment channel networks for other cryptocurrencies using similar primitives (e.g., Ethereum's Raiden Network [10]).

An important problem for a payment channel network is how to choose routes for transactions. In the Lightning Network, each node keeps a routing table for the rest of the network and source-routes transactions [143]. The Lightning network nodes run a link-state routing protocol [47] to build their routing tables: they broadcast information about payment channels to the network, enabling each node to maintain a local view of the network topology. The Lightning Network specification does not state how nodes should pick routes using the topology; current implementations use simple

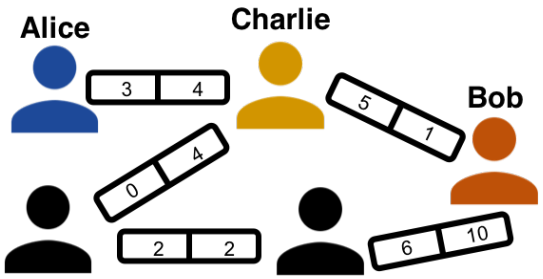


Figure 6.2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

shortest path algorithms. Additionally, the Lightning Network uses onion routing [85] to provide privacy. Specifically, onion routing ensures that intermediate routers cannot determine the source and destination of a payment.

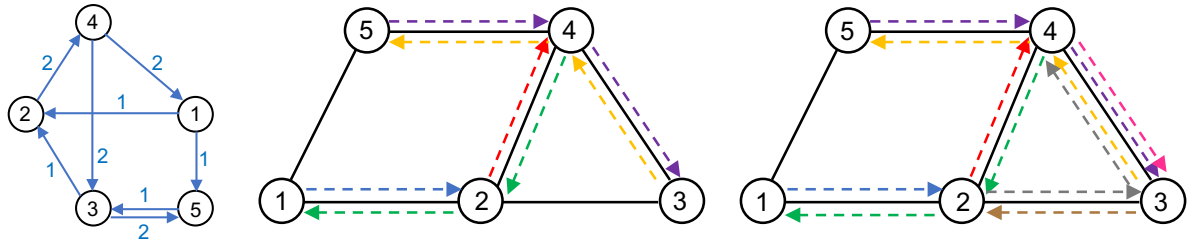
The routing problem has recently begun to receive some attention in the research literature. A key benchmark is the *max-flow* routing algorithm [76]. For each transaction, max-flow uses a distributed implementation of the Ford-Fulkerson method to find source-destination paths that support the largest transaction volume. If this volume exceeds the transaction value, the transaction succeeds. Max-flow routing is often considered the gold standard in terms of throughput and transaction success rate, but it has high overhead, requiring $O(|V| \cdot |E|^2)$ computation per transaction, where $|V|$ and $|E|$ are the number of nodes and edges in the network, respectively [184].

Two main alternatives have been proposed: landmark routing and embedding-based routing. In *landmark routing*, select routers (landmarks) store routing tables for the rest of the network; hence individual nodes only need to route transactions to a landmark [174]. This approach is used in systems like Flare [145] and SilentWhispers [119, 128]. *Embedding-based* or *distance-based* routing instead learns a vector embedding for each node, such that nodes that are close in network hop distance are also close in embedded space. Each node relays each transaction to the neighbor whose embedding is closest to the destination's embedding. Systems like VOUTE [152] and SpeedyMurmurs [153] use embedding-based routing. One challenge is computing and updating the embedding as the graph and link balances change over time.

6.3 Imbalance-Aware Routing

The key to effective routing in payment channel networks is to keep payment channels *balanced*. A payment channel becomes imbalanced when the transaction rate across it is higher in one direction than the other; the party making more payments eventually runs out of funds and cannot send further payments until it either receives funds from the other side, or it deposits new funds into the payment channel via an on-chain transaction. Since on-chain transactions are expensive and slow, it is desirable to avoid them as much as possible.

In this section, we take a first-principles approach to routing with rate-imbalance constraints. We first answer the question: What is the maximum achievable throughput in a payment channel network with and without on-chain rebalancing transactions? We use a fluid model of the network, wherein transactions between source, destination pairs are modeled as continuous flows, to show that the maximum achievable throughput depends on properties of a *payment graph* that captures how



(a) Payment graph (b) Shortest-path routing (c) Optimal balanced routing

Figure 6.3: Example illustrating balanced routing. (a) The payment graph shows the desired transaction rates between pairs of nodes. (b) and (c) The maximum transaction rate achievable with shortest-path routing and the optimal balanced routing schemes on the topology shown. Each dotted edge represents 1 unit of flow along the direction of the arrow. The colors indicate separate flows.

currency flows between network participants in Section 6.3.2.2. We formulate optimization problems for routing with rate-imbalance constraints. These optimization problems generalize classical network utility maximization [169] formulations for routing and rate control in communication networks [98, 62]. We discuss how dual decomposition of these optimization problems leads naturally to decentralized algorithms for imbalance-aware routing in payment channel networks in Section 6.3.3.

6.3.1 A Motivating Example

To illustrate the importance of balanced routing, consider the 5-node payment-channel network shown in Fig. 6.3. Suppose sender, receiver pairs seek to transact at the rates shown in Fig. 6.3a. For example, node 1 wishes to send at rate 1 to nodes 2 and 5, and node 2 wishes to send at rate 2 to node 4. Fig. 6.3 shows two different routing strategies under these demands for a specific network topology. In each case, we impose the constraint that the net rate in either direction of any edge must be equal to ensure that payment channels do not run out of funds.

Fig. 6.3b shows the result for shortest-path balanced routing, wherein senders route only along the shortest path to their receiver nodes. For example, node 4 routes a flow of rate 1 along the path $4 \rightarrow 2 \rightarrow 1$ (shown in green). The maximum total rate (*throughput*) that can be sent by this routing scheme is 5 units; any rate assignment offering a higher throughput does not satisfy the rate-balance constraints. However, an alternate routing scheme of Fig. 6.3c in which senders do not necessarily send along the shortest paths achieves a higher throughput of 8 units. Here, node 2 sends a flow of rate 1 along the path $2 \rightarrow 3 \rightarrow 4$, while the shortest path is $2 \rightarrow 4$. This enables nodes 3 and 4 to also send 1 unit of flow to nodes 2 and 3 respectively.

6.3.2 Limits on Throughput

6.3.2.1 Fluid Model

Consider the payment channel network modeled as a graph $G(V, E)$, with routers V and payment channels E. For any source, destination routers $i, j \in V$, let $d_{i,j} \geq 0$ denote the average rate at which transaction units have to be transferred from i to j. Let c_e denote the total amount of funds in channel e, for $e \in E$, and Δ the average latency experienced by transactions due to network delays. Lastly, let $\mathcal{P}_{i,j}$ denote the set of paths from i to j in G, for $i, j \in V$. We include only 'trails', i.e., paths without repeated edges, in $\mathcal{P}_{i,j}$.

The network attempts to satisfy the demands $d_{i,j}$ by sending *flows* from source to destination routers. A flow is defined by a (path, value) tuple, where path denotes the route taken and value denotes the rate carried by the flow. Operationally this can be interpreted as routing transactions along the flow's path such that the average rate of currency transferred along the path equals the value of the flow.

Valid flows must obey two main constraints. First, payment channels are limited in their capacity to process a large rate of transactions, because funds allocated to a transaction are unavailable for use in other transactions for some amount of time. Assume it takes Δ seconds on average for a transaction to reach its destination, get confirmed, and its hash lock key to reach a payment channel on the upstream path. Then a payment channel with c units of capacity can support, on an average, transactions of net value no more than c/Δ currency units per second. Second, to maintain balance the total flow along the two directions of a payment channel must be the same. If the flows are not balanced, the channel will eventually become unusable along one of the directions unless it is periodically rebalanced via on-chain transactions (we revisit this possibility below).

With these constraints, maximizing throughput in the fluid model is equivalent to finding flows of maximum total value and can be formulated as a Linear Program (LP):

$$\text{maximize} \quad \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p \tag{6.1}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \tag{6.2}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{6.3}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq 0 \quad \forall (u,v) \in E \tag{6.4}$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P}, \tag{6.5}$$

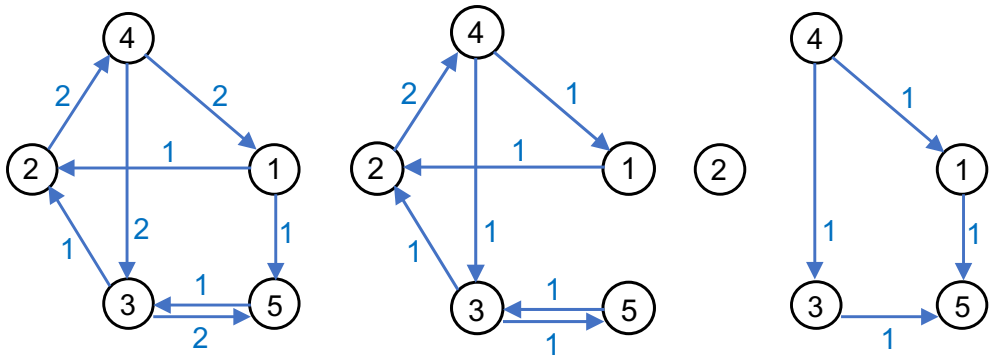
Here, x_p denotes the flow along path p and $\mathcal{P} = \cup_{i,j \in V} \mathcal{P}_{i,j}$ is the set of all paths. The constraints in eq. (6.3) reflect the capacity limits of payment channels, and the constraints in eq. (6.4) enforce the balance requirement.

6.3.2.2 Throughput under Perfect Balance

Next, we show that the maximum throughput achievable under perfect balance is fundamentally restricted by the structure of demand $[d]_{i,j}$ across nodes. Specifically the maximum throughput possible is equal to the largest amount of "circulation" contained in the demands, with higher throughputs possible only if payment channels are rebalanced via on chain transactions.

Payment graphs and circulation. Define a *payment graph* $H(V, E_H)$ as a weighted directed graph with nodes V and edges E_H. An edge $(i,j) \in E_H$ if $d_{i,j} > 0$ with $d_{i,j}$ also being the weight of that edge. Payment graphs do not depend on the network topology; they depend only on the pattern of payments between the network nodes. Payment graphs are useful for analyzing throughput because any flow imbalance across cuts of H cannot be balanced by any routing scheme in the payment channel network G. Fig. 6.4a shows the payment graph for the example discussed in Section 6.3.1.

We define the *circulation* graph of a payment graph H as another directed weighted graph $C(V, E_C)$ with $E_C \subseteq E_H$, $w_C(i,j) \leq w_H(i,j)$ for all $(i,j) \in E_C$, and the total weight of incoming and outgoing edges being equal at any node. $w_C(i,j)$, $w_H(i,j)$ denote edge weight of edge (i,j) in



(a) Payment graph (b) Circulation (c) DAG

Figure 6.4: Payment graph corresponding to the demands in the example of Fig. 6.3. It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

graphs C, H respectively. The circulation graph captures flows that form cycles in the payment graph. Letting $\sum_{(i,j)\in E_C} w_C(i,j)$ be the total *value* $\nu(C)$ of the circulation, there exists a circulation graph C^* of maximum value for a given payment graph H. Maximum circulation graphs are not necessarily unique. A maximum circulation graph C^* can be constructed by successively removing cycles of constant flow from H, and adding them to C^*. The graph remaining after removing all cycles from H is a weighted directed acyclic graph (DAG). Figures 6.4b and 6.4c show the decomposition of the payment graph of Fig. 6.4a into circulation and DAG components.

Proposition 1. *For a payment graph H with a maximum circulation graph C^*, there exists a routing scheme that achieves a throughput of $\nu(C^*)$ with perfect balance (i.e., $B = 0$) on a network with payment channels of unlimited capacity. Conversely, no routing scheme can achieve a throughput greater than $\nu(C^*)$ with perfect balance on any network.*

Proof. To see that a throughput of $\nu(C^*)$ is achievable, consider routing the circulation along any spanning tree T of the payment network G. For any pair of nodes $i, j \in V$ there exists a unique path from i to j in T through which $w_{C^*}(i,j)$ amount of flow can be routed. We claim that such a routing scheme is perfectly balanced on all the links. This is because for any partition $S, V\backslash S$ of C^*, the net flow going from S to $V\backslash S$ is equal to the net flow going from $V\backslash S$ to S in C^*. Since the flows along an edge e of T correspond precisely to the net flows across the partitions obtained by removing e in T, it follows that the flows on e are balanced as well.

Next, to see that no balanced routing scheme can achieve a throughput greater than $\nu(C^*)$, assume the contrary and suppose there exists a balanced routing scheme SCH with a throughput greater than $\nu(C^*)$. Let $H_{\texttt{SCH}} \subseteq H$ be a payment graph where the edges represent the portion of demand that is actually routed in SCH. Since $\nu(H_{\texttt{SCH}}) > \nu(C^*)$, $H_{\texttt{SCH}}$ is not a circulation and there exists a partition $S, V\backslash S$ such that the net flow from S to $V\backslash S$ is strictly greater than the net flow from $V\backslash S$ to S in $H_{\texttt{SCH}}$. However, the net flows routed by SCH across the same partition $S, V\backslash S$ in G are balanced (by assumption) resulting in a contradiction. Thus we conclude there does not exist any balanced routing scheme that can achieve a throughput greater than $\nu(C^*)$. □

Proposition 1 shows that the maximum achievable throughput in a payment channel network with perfect balance can be less than 100% if the demands for payments between nodes do not define a circulation. For example, the routing presented in Fig. 6.3c corresponds precisely to routing the maximum circulation component of the payment graph (Fig. 6.4b) and is hence optimal. Yet it is

only able to route $8/12 = 75\%$ of the demands in the payment graph. The DAG component is not routable without violating the balance constraints.

Using payment channels with higher capacity can mitigate problems caused by rate imbalance, but it cannot solve it fundamentally. A payment channel with more funds can sustain rate imbalance for a longer period of time, but it will eventually run out of funds if the rate imbalance persists. To remain usable in both directions, such a payment channel will have to be repeatedly rebalanced via on-chain transactions. In current implementations, rebalancing requires closing a payment channel and reopening a new one. However, two nodes can in principle use multiple smaller payment channels instead of a single large payment channel, so that they can rebalance them one at a time.

We next consider the possibility of such on-chain rebalancing and analyze its impact on throughput in the fluid model.

6.3.2.3 Throughput with On-Chain Rebalancing

We model on-chain payment channel rebalancing using variables $b_{(u,v)}$, which is the average rate at which payment channel (u, v) receives new funds in the $u \to v$ direction via on-chain transactions. These funds would typically be withdrawn from another payment channel on which node u is receiving more than it is sending. A positive rebalancing rate on some payment channels ($b_{(u,v)} > 0$) can improve network throughput beyond the upper bound presented in Proposition 1. However, rebalancing channels on chain is expensive for the routers both in time (due to transaction confirmation delays) and in transaction fees (paid to the miners). Thus routers may be unwilling to actively rebalance their payment channels unless the profit obtained from the higher throughput is sufficient to offset the rebalancing costs.

We leave a full analysis of incentives for routers to future work, but we now consider a simple modification to the previous optimization problem to understand the impact of on-chain rebalancing:

$$\text{maximize} \quad \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p - \gamma \sum_{(u,v) \in E} b_{(u,v)} \tag{6.6}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \tag{6.7}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{6.8}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \tag{6.9}$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \tag{6.10}$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \tag{6.11}$$

The objective function in the problem above consists of two terms. The first term is the total throughput, which we would like to maximize. The second term is the "cost" of on-chain rebalancing at a total rate of $\sum_{(u,v) \in E} b_{(u,v)}$, which we would like to minimize. The parameter γ dictates how we weigh these two conflicting goals. We can interpret γ as the increase in throughput required to offset one unit of on-chain rebalancing rate. In practice, γ would depend on a variety of factors, such as the ratio of blockchain transaction fees to routing fees and how the transaction fees are computed. For example, if blockchain transaction fees do not depend on transaction size, routers can reduce the cost of rebalancing by using larger transactions, thereby effectively reducing γ in the above model.

We now analyze how on-chain rebalancing impacts throughput. Let $B \triangleq \sum_{(u,v) \in E} b_{(u,v)}$ be the total rate of on-chain rebalancing, and let's assume for simplicity that the payment channel

capacities are unlimited. For large γ, B will be close to 0 in the optimal solution. From Proposition 1, we know that the maximum achievable throughput in this case is $\nu(C^*)$. On the other hand, for $\gamma \approx 0$, B can become arbitrarily large. It is not difficult to see that the network throughput can satisfy all of the demand $\sum_{i,j} d_{i,j}$ in this case, provided that the links have sufficient capacity (i.e, if the constraints (6.8) are not tight). In general, as γ decreases, the total throughput and total rebalancing rate both increase in the optimal solution, until the throughput reaches the maximum possible throughput given the capacity constraints.

We can show that the maximum achievable throughput is a non-decreasing concave function of the total rebalancing rate. The argument is as follows. Let $t(B)$ be the maximum achievable throughput for a bound B on the total rebalancing rate. $t(B)$ is the solution of the following optimization problem:

$$\text{maximize} \quad \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p \tag{6.12}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \tag{6.13}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{6.14}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \tag{6.15}$$

$$\sum_{(u,v) \in E} b_{(u,v)} \leq B \tag{6.16}$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \tag{6.17}$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \tag{6.18}$$

$t(\cdot)$ is non-decreasing because the set of feasible solutions is non-decreasing in B. To see that $t(\cdot)$ is concave, consider arbitrary B_1 and B_2 and $\theta \in (0,1)$. Let $\{(x_p^1, b_{(u,v)}^1) : p \in \mathcal{P}, (u,v) \in E\}$ and $\{(x_p^2, b_{(u,v)}^2) : p \in \mathcal{P}, (u,v) \in E\}$ be the optimal solutions corresponding to B_1 and B_2. Then $\{(\theta x_p^1 + (1-\theta)x_p^2, \theta b_{(u,v)}^1 + (1-\theta) b_{(u,v)}^2) : p \in \mathcal{P}, (u,v) \in E\}$ is a feasible solution for bound $\theta B_1 + (1-\theta)B_2$, and it achieves a throughput of $\theta t(B_1) + (1-\theta)t(B_2)$. Hence, $t(\theta B_1 + (1-\theta)B_2) \geq \theta t(B_1) + (1-\theta)t(B_2)$, and $t(\cdot)$ is concave.

6.3.3 Algorithms

In this section, we derive decentralized algorithms for routing and rate control in payment channel networks based on the previous optimization problems. These algorithms follow naturally from a dual decomposition of the optimization problems and have the following basic structure. Each payment channel has a *price* in each direction. Routers locally update these prices based on both congestion and imbalance across payment channels. To select paths, end-hosts monitor the total price of different paths and choose the cheapest options. Using dual variables for prices is common in the utility-maximization-based rate control and routing literature (e.g., see [98]). A key difference from prior work, however, is that in addition to price variables for link capacity constraints, we also have price variables for link balance constraints. This ensures that links with skewed balances have a high price and helps steer the routing towards rebalancing the payment channels.

We focus on the optimization problem in eqs. (6.6)–(6.11) for the general case with on-chain rebalancing. The algorithm without on-chain rebalancing is a special case.

Consider the partial lagrangian:

$$L(\mathbf{x},\mathbf{b},\lambda,\mu) = \sum_{i,j\in V}\sum_{p\in\mathcal{P}_{i,j}} x_p - \gamma \sum_{(u,v)\in E} b_{(u,v)} - \sum_{(u,v)\in E} \lambda_{(u,v)} \left[\sum_{p\in\mathcal{P}:(u,v)\in p} x_p + \sum_{p\in\mathcal{P}:(v,u)\in p} x_p - \frac{c_{(u,v)}}{\Delta} \right] - \sum_{(u,v)\in E} \mu_{(u,v)} \left[\sum_{p\in\mathcal{P}:(u,v)\in p} x_p - \sum_{p\in\mathcal{P}:(v,u)\in p} x_p - b_{(u,v)} \right], \quad (6.19)$$

where $\lambda_{(u,v)}$ and $\mu_{(u,v)}$ are Lagrange variables corresponding to the capacity and rate-imbalance constraints in eq. (6.8) and eq. (6.9) respectively. The partial Lagrangian can be rewritten as:

$$L(\mathbf{x},\mathbf{b},\lambda,\mu) = \sum_{i,j\in V} \left[\sum_{p\in\mathcal{P}_{i,j}} x_p - \sum_{p\in\mathcal{P}_{i,j}} x_p \sum_{(u,v)\in p} \left(\lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)}\right) \right] + \sum_{(u,v)\in E} \left(\mu_{(u,v)} - \gamma\right) b_{(u,v)} + \sum_{(u,v)\in E} \lambda_{(u,v)} \frac{c_{(u,v)}}{\Delta}. \quad (6.20)$$

Define $z_{(u,v)} \triangleq \lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)}$ to be the price for edge $(u,v) \in E$, and $z_p \triangleq \sum_{(u,v)\in p} z_{(u,v)}$ to be price for path $p \in \mathcal{P}$. The important fact about the above partial lagrangian is that it decomposes into separate terms for the rate variables of each source/destination pair $\{x_p : p \in \mathcal{P}_{i,j}\}$, and the rebalancing rate of each edge $b_{(u,v)}$. This suggests the following primal-dual algorithm for solving the optimization problem in eqs. (6.6)–(6.11):

- **Primal step.** Given the edge prices $z_{(u,v)}(t)$ at time t, for each source/destination pair (i,j), update the sending rate on paths $p \in \mathcal{P}_{(i,j)}$ as follows:

$$x_p(t) = x_p(t-1) + \alpha\left(1 - z_p(t)\right), \quad x_p(t) = \mathrm{Proj}_{\mathcal{X}_{i,j}}(x_p(t)), \quad (6.21)$$

  where $\mathrm{Proj}_{\mathcal{X}_{i,j}}(\cdot)$ stands for projection onto the convex set $\mathcal{X}_{i,j} = \{x_p \text{ for } p \in \mathcal{P}_{i,j} | x_p \geq 0, \sum_{p\in\mathcal{P}_{i,j}} x_p \leq d_{i,j}\}$.

  At time t, each edge (u,v) also updates its on-chain rebalancing rate according to:

$$b_{(u,v)}(t) = \left[b_{(u,v)}(t-1) + \beta\left(\mu_{(u,v)}(t) - \gamma\right)\right]_+, \quad (6.22)$$

  where $[\cdot]_+ \triangleq max(\cdot, 0)$.

- **Dual step.** In the dual step, the routers update the edge prices based on capacity and imbalance constraints at the payment channels. Specifically, each edge performs the following updates independently:

$$\lambda_{(u,v)}(t+1) = \left[\lambda_{(u,v)}(t) + \eta\left(\sum_{p\in\mathcal{P}:(u,v)\in p} x_p(t) + \sum_{p\in\mathcal{P}:(v,u)\in p} x_p(t) - \frac{c_{(u,v)}}{\Delta}\right)\right]_+, \quad (6.23)$$

$$\mu_{(u,v)}(t+1) = \left[\mu_{(u,v)}(t) + \kappa\left(\sum_{p\in\mathcal{P}:(u,v)\in p} x_p(t) - \sum_{p\in\mathcal{P}:(v,u)\in p} x_p(t) - b_{(u,v)}\right)\right]_+. \quad (6.24)$$

The parameters $\alpha, \beta, \eta, \kappa$ are positive "step size" constants that determine the speed of the dynamics. Using standard arguments, it can be shown that for sufficiently small step sizes, the above algorithm converges to the optimal solution of the optimization problem in eqs. (6.6)–(6.11).

The algorithm has the following intuitive interpretation: $\lambda_{(u,v)}$ and $\mu_{(u,v)}$ are prices that vary due to capacity constraints and imbalance at the payment channels. In eq. (6.23), $\lambda_{(u,v)}$ increases if the net rate across the payment channel (u, v) (in both directions) exceeds its capacity, and it decreases down to zero if there is excess capacity. Similarly, in eq. (6.24), $\mu_{(u,v)}$ increases if the rate in the (u, v) direction exceeds the rate in the (v, u) direction by more than $b_{(u,v)}$, the rate at which funds are deposited for edge (u, v) on chain. As these prices vary, each source/destination pair reacts by changing the sending rate on its paths according to eq. (6.21). The effect is to reduce the rate on expensive paths and increase it on cheap paths. Simultaneously, each edge also adapts its on-chain rebalancing rate based on eq. (6.22). If $\mu_{(u,v)} > \gamma$, the price charged for imbalance is higher than the cost of on-chain rebalancing per unit rate. Hence the edge increases its on-chain rebalancing rate $b_{(u,v)}$; otherwise it decreases it.

Finally, we note that in the absence of any on-chain rebalancing, the algorithm can be simplified by setting $b_{(u,v)} = 0$ for all edges (u, v).

6.3.3.1 Practical considerations

In practice, routers have to dynamically estimate the rate over their payment channels from the transactions that they encounter. The source nodes, whenever they have to send transactions, query for the path prices, and adapt the rate on each path based on these prices. This rate adaptation can occur in different ways depending on the implementation. For example, in an implementation where source nodes use only one path for routing each transaction, they can select these paths such that the frequency of usage of different paths over time is roughly proportional to the optimal flow rate along the paths. In the next section, we describe a design that allows more fine-grained rate control by splitting transactions into small units that can be transmitted on different paths.

A typical challenge with such iterative algorithms for adjusting path prices and sending rates is slow convergence. An algorithm that is slow to converge may not be able to adapt the routing to changes in the transaction arrival pattern. If the transaction arrival patterns change frequently, this may result in a perpetually suboptimal routing.

Therefore, in practice, it may be beneficial to also consider simpler approaches to balancing transactions across different paths that can converge quickly. One such approach is for sources to independently try to minimize imbalance on their paths by always sending on paths with the largest available capacity, much like "waterfilling" algorithms for max-min fairness. A source measures the available capacity on a set of paths to the destination. It then first transmits on the path with highest capacity until its capacity is the same as the second-highest-capacity path; then it transmits on both of these paths until they reach the capacity of the third highest-capacity-path, and so on.

For both types of algorithms, practical implementations would restrict the set of paths considered between each source and destination, so that the overhead of probing the path conditions is not too high. There are a variety of possible strategies of selecting these paths, e.g., the K shortest paths or the K highest-capacity paths between every source/destination pair. We leave an investigation of the best way to select the paths to future work.

6.4 The Spider Network

In current payment channel networks, the sender first finds one or more paths with enough funds ("capacity") to fully satisfy the payment, and only then transmits it by sending one transaction on

each path. This approach is similar to circuit switching and has several drawbacks. First, it makes it difficult to support large payments. If the sender cannot find paths to send the payment in one shot, the payment fails. Second, it exacerbates imbalance on payment channels, since a large transaction can deplete funds on one side of a payment channel. As discussed previously, the party that runs out of funds cannot send more payments until it either receives payments from the other side, or it replenishes funds via a transaction on the blockchain. The result is *head of line blocking*, where large transactions can block shorter payments that could have been serviced quickly.

Spider is a packet-switched payment channel network that solves these problems. At a high-level, Spider hosts send payments over the network by transmitting a series of *transaction units*, much like packets in a data network. Each transaction unit transfers an amount of money bounded by the *maximum transaction unit (MTU)*. Transaction units from different payments are queued at Spider routers, which transmit them as funds become available in payment channels.

6.4.1 Spider Hosts

Spider hosts run a transport layer that provides standard interfaces for applications to send and receive payments on the network. We envision a message-oriented transport rather than a stream-oriented transport like TCP. To send a payment, the application specifies the destination address, the amount to send, a deadline, and the maximum acceptable routing fee.

The transport provides interfaces for both atomic and non-atomic payments. It guarantees that atomic payments are either fully delivered or that no payment takes place. For non-atomic payments, the transport is allowed to partially deliver the payment; it must then inform the sender precisely how much it delivered by the deadline and ensure that no further transactions are made as part of that payment. The sender can attempt the remainder of the payment at a later time, or decide to complete the payment on the blockchain. As we will see, relaxing atomicity improves network efficiency, and we therefore expect the routing cost for non-atomic payments to be cheaper.

Non-atomic payments. Recall that transactions are locked by a cryptographic hash lock, whose private key is known only to the sender (S6.1). In Spider, the sender generates a new key for every transaction unit. To implement non-atomic payments, the sender simply waits for confirmation from the receiver that she has received a transaction unit (identified by a payment ID and sequence number), and only then sends her the key. The sender therefore knows exactly how much of a payment the receiver can unlock. It can withhold the key for "in-flight" transactions that arrive after the deadline, or proactively cancel them by informing the routers.

Atomic payments. Spider is also compatible with atomic payments using recently-proposed mechanisms like Atomic Multi-Path Payments (AMP) [1] that split a payment over multiple paths while guaranteeing atomicity. The idea is to derive the keys for all the transaction units of a payment from a single "base key," and use additive secret sharing to ensure that the receiver cannot unlock any of the transaction units until she has received all of them.

Congestion control. Spider hosts use a congestion control algorithm to determine the rate to send transaction units for different payments. Designing congestion control algorithms for payment channel networks is beyond the scope of this chapter, but we briefly remark on some interesting aspects. Standard goals for congestion control in data networks such as high utilization, fairness, and low delay also apply to payment channel networks. Additionally, transfers in payment channel networks have deadlines, and therefore approaches that adapt congestion control to meet deadlines are particularly relevant [175, 93]. To make congestion control decisions, hosts can use implicit signals like delay or explicit signals from the routers (e.g., queue sizes, available capacity, imbalance, etc.).

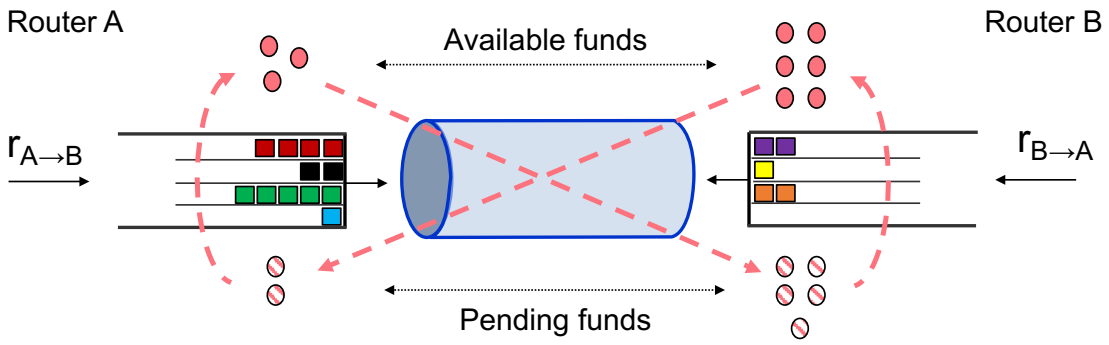


Figure 6.5: Routers queue transaction units and schedule them across the payment channel based on available capacity and transaction priorities. Funds received on a payment channel remain in a pending state until the final receiver provides the key for the hash lock.

A unique aspect of payment channel networks is that sending at higher rates does not always reduce capacity for other payments, but may in fact improve performance for other payments by promoting balance across payment channels. For example, if a sender discovers that payment channels on certain paths have a high imbalance in the downstream direction, it may aggressively increase its rate to balance those channels. Exploring imbalance-aware congestion control algorithms is an interesting direction for future work.

6.4.2 Spider Routers

Spider routers are responsible for forwarding transaction units to the intended receiver. Existing designs like the Lightning Network use Onion routing [85] to ensure privacy of user payments. Spider routers can use similar mechanisms for each transaction unit to provide privacy [7].

A Spider router queues transaction units when it lacks the funds to send them immediately (Fig. 6.5). As it receives funds from the other side of the payment channel, it uses them to send new transaction units from its queue. Funds from new-arrived transaction units are not available to use immediately. The router must wait until it receives the key for the hash lock from the final destination. This delay limits the capacity of a payment channel. If a transaction takes on average Δ seconds to confirm, then a payment channel with total funds c can support, on average, transactions of net value no more than c/Δ currency units per second.

An important benefit of Spider's packet-switched architecture is that the routers can schedule transaction units based on payment requirements. For example, they can prioritize payments based on size, deadline, or routing fees [18].

6.5 Preliminary Evaluation

6.5.1 Setup

Simulator. We modified an existing simulator for payment channel networks [12] to model transaction arrivals and completion events. Arriving transactions are routed according to the routing algorithm as long as funds are available on the paths chosen by the algorithm. Payments that are actually routed incur a delay of 0.5 seconds before the funds are available at the receiver. In the meantime, these funds are held *inflight* and are unavailable for use by any party along the path. As and when a transaction completes, these funds are released. The simulator supports non-atomic

payments through a global queue that tracks all incomplete payments. These transactions are periodically polled to see if they can make any further progress. They are then scheduled according to a scheduling algorithm. We leave implementing in-network queues and rate control to future work.

Dataset. We evaluated the algorithms on two different topologies: an ISP-topology [5] and a subgraph from the original topology of Ripple [11], an existing currency exchange network that transacts in XRPs. The ISP topology is relatively simple allowing us to reason about the dynamics of the system. We used a graph with 32 nodes and 152 edges and generated $200,000$ transactions to send on it. The transactions were synthetically generated with the sizes sampled from Ripple data after pruning out the largest 10%. The average transaction size for this dataset is 170 XRP with the largest one being 1780 XRP. The sender for each transaction was sampled from the set of nodes using an exponential distribution while the receiver was sampled uniformly at random. We set all edges in the graph to have the same capacity, which we varied from 10000 XRP to 100000 XRP per link in different experiments.

We also used data from the Ripple network from January 2013 [12]. The original dataset had 90,000 nodes and 330,000 edges. We pruned the dataset to remove the degree-1 nodes (which don't make routing decisions) as well as edges with no funds between them. The largest resulting component had 3774 nodes and 12512 edges. The $75,000$ transactions from the original dataset that are between nodes in this subgraph have an average size of 345 XRP with the largest transaction size being 2892 XRP. Consequently, we set the capacity of all the links in the reduced Ripple graph to 30000.

Schemes. We evaluate SpeedyMurmurs [153], SilentWhispers [119], and max-flow routing. All of these schemes use atomic payments. We implemented shortest-path routing with non-atomic payments as another baseline for our packet-switched network. We compared these schemes to Spider (LP) and Spider (Waterfilling), the routing algorithms described in S6.3. Spider (LP) solves the LP in Eq. (6.1) once based on the long-term payment demands and uses the solution to set a weight for selecting each path. We restrict both algorithms to use 4 disjoint shortest paths for every source-destination pair. All non-atomic payments are scheduled in order of increasing incomplete payment amount, i.e. according to the *shortest remaining processing time (SRPT)* policy [18].

Metrics. We evaluate these routing schemes for their *success ratio* and *success volume*. The former captures how many payments amongst those tried actually completed. The latter focuses on the volume of payments that went through as a fraction of the total volume across all attempted payments.

6.5.2 Results

We summarize our results in Fig. 6.6. The results were collected at the end of $200s$ for the ISP topology and $85s$ for the Ripple topology. All the edges in both topologies were initialized with a capacity of 30000 XRP, equally split between the two parties. We can see that splitting the payments into transaction units and scheduling them according to SRPT already provides a 10% increase in success ratio over SpeedyMurmurs and SilentWhispers even for the shortest path routing scheme. Although Max-flow performs quite well, it has a high overhead per transaction as discussed in Section 6.2. In comparison, Spider (Waterfilling) is able to leverage knowledge of imbalance to perform within 5% of Max-flow despite being restricted to only 4 paths.

Spider (LP), on the other hand, attains a success volume of 52% and 22% for the ISP and Ripple topologies respectively. Both of these correspond precisely to the circulation component of the payment graph. This is because Spider (LP) uses an estimate of the demand matrix to make decisions for the entire duration of the simulation. While this approach works for a stationary

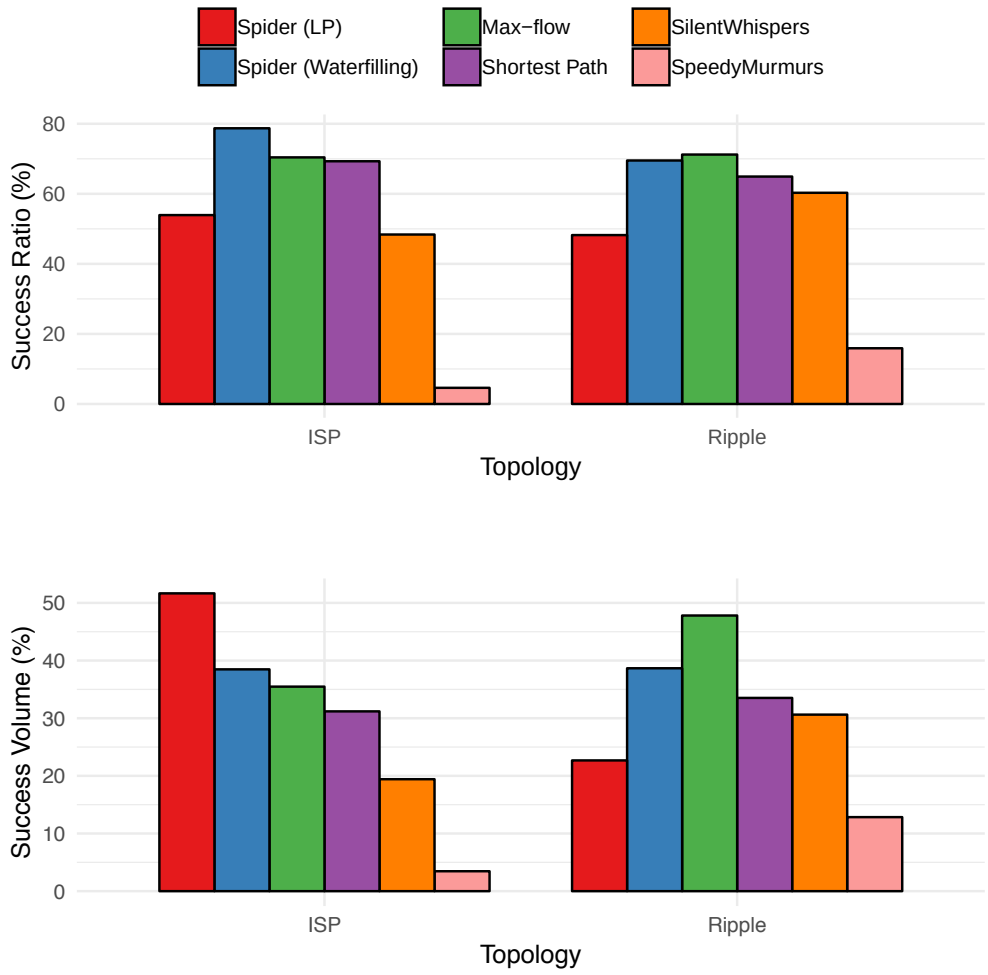


Figure 6.6: Comparison of payments completed across schemes on the ISP and Ripple Topologies when the capacity per link is 30,000

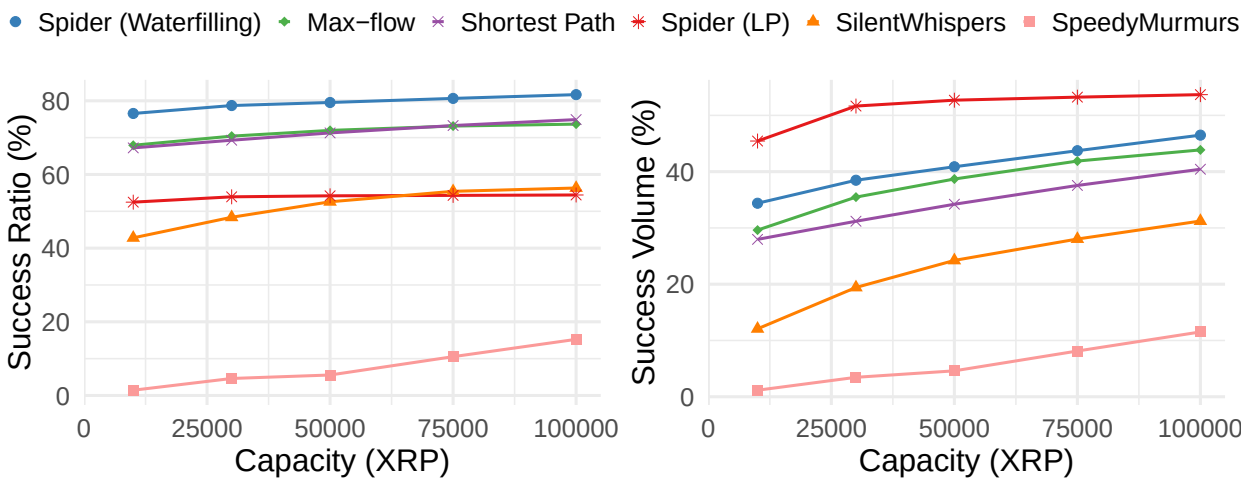


Figure 6.7: Effect of increasing capacity per link on the success metrics when routing payments on the ISP topology. All links in the network have the same credit.

transaction arrival pattern, as is the case with the ISP topology, it does not work too well for the Ripple network in which the traffic demands vary over time. Further, the LP assigns zero flows to all paths for certain commodities which means no payments between them will ever get attempted. We plan to explore other objectives like proportional fairness [98] in the future to overcome this problem.

How does capacity impact success? We varied the capacity on each link in the ISP topology from 10000 XRP to 100000 XRP and measured the success across the schemes. Fig. 6.7 summarizes the results. As expected, as the capacity increases, more transactions start succeeding. The total volume of successful transactions also experiences an increase. Additionally, to achieve a certain success volume or success ratio, the amount of capital that needs to be locked in with Spider (Waterfilling) is much lower than what would need to be invested in any other scheme. Spider (LP) is less sensitive to changes in capacity, because it does a better job of avoiding imbalance.

6.6 Discussion and Future Work

This chapter proposes a new packet-switched architecture for payment channel networks. A first step in sending transaction units through this network is the discovery of payment paths. To that effect, we explored the benefits of imbalance-aware routing approaches in this paper. However, we envision more potential gains from collaboration between routers and end-hosts. Spider routers have information about transaction units belonging to payments across many sender-receiver pairs. This can be leveraged for admission control. In other words, routers can decide payment priorities or reject some extremely large transactions that are unlikely to complete within the deadline. On the other hand, Spider end hosts could employ rate control strategies and their own admission control policies influenced by feedback on payment completion rates or router hints.

Our routing algorithms suggest a way to set routing fees to maximize network throughput with rational users that prefer cheaper routes. However, our design does not address incentives for network service providers that wish to maximize their profits from routing fees. Further, we have assumed routers do not inject additional funds into the payment channels after their initial escrow; however routers have an incentive to periodically replenish funds if it leads to an overall increase in their return on investment from routing fees. An analysis of economic incentives for network stakeholders

is an exciting direction for future work.

Lastly, we have focused on optimizing payment delivery for a given network topology and payment channel capacities. However, our results show fundamental limits to performance based on the flow of currency across the network. Understanding how best to design network topology and influence its formation in a decentralized payment channel network is an important area for investigation.

6.7 Acknowledgement

The authors thank V. Sivaraman and S. Venkatakrishnan for collaborations on the design of `Spider`. This chapter includes and expands on material in [165].

Chapter 7

Economics

Giulia Fanti, CMU
Leonid Kogan, MIT
Sewoong Oh, UW Seattle
Pramod Viswanath, UIUC

The economic properties of cryptocurrencies are of central interest to a payment system and the focus of this chapter. Economic properties are closely intertwined with the specifics of the system design; this is particularly true for proof-of-stake (PoS) cryptocurrencies. In this chapter, we focus on three broad and connected issues related to the design of Unit-e: (i) valuation of cryptocurrencies in the proof-of-stake setting, (ii) the selection of block rewards, and (iii) the selection and disbursal of transaction fees. In particular, we highlight the delicate connection between these issues and our overall design of Unit-e as a proof-of-stake (PoS) payment system.

This chapter has three contributions:

1. We present a theoretical valuation framework for PoS payment systems. Although cryptocurrencies can have applications beyond just payment processing, this framework gives us a concrete lower bound for reasoning about the fiat valuation of Unit-e. Unlike PoW cryptocurrencies, the first-order effect is no longer the cost of keeping the network running (i.e., mining). Instead, token valuation stems from a combination of two factors: consumer demand for holding tokens, and validator demand for holding tokens. The first term is inversely proportional to token velocity, whereas the second term is proportional to the economic value of the aggregate transaction fee flow. Our analysis provides an important input into various design decisions in PoS systems, including monetary policy, rewards, transaction fees, and network security mechanisms.

2. We study the effects of block rewards on user adoption and equity of rewards among block proposers and validators. One key insight is that in PoS systems, severe stake imbalance can arise from compounding in users' stake pools, even if nodes are following protocol. That is, a node that starts with a small fraction of stake can end up with an arbitrarily high fraction of stake, simply due to randomness in the PoS proposal stage. We show theoretically that this effect can be mitigated through two factors: choice of block reward function and size of the initial stake pool. We prove the optimality (in terms of maximizing equitability) of a novel *geometric block reward function* and advocate for starting with a large initial stake pool to mitigate the effects of stake compounding.

3. We discuss various transaction fee mechanisms, highlighting key tradeoffs of various design choices in terms of supporting low congestion and stable valuations according to the models in part (1).

We want to emphasize that valuation and incentive mechanisms are closely intertwined. Incentive mechanisms play a dual role in the payment network: they elicit "good" behavior from network participants, such as block validators; they also distribute rewards, thereby affecting the allocation of tokens and the overall valuation. Our focus here is on the second aspect. Because tokens can be easily re-allocated across the network, they must earn the same rate of return regardless of how they are used. This implies that a system's choice of incentive mechanisms in relation to any particular activity (e.g., block validation, or off-chain transaction routing) potentially affects the entire system through its effect on token valuation. For example, suppose one were to increase rewards for block validation: each token staked would now earn a higher expected number of tokens per unit of time as a reward. This would attract tokens towards validation and away from other activities, e.g., from off-chain transaction routing. The result would be increased congestion for on-chain transactions, and lower transaction volume of off-chain transactions. Another implication could be a reduction in the rate of growth of the token price, which would act to offset the increased level of rewards for validation in terms of tokens. This, in turn, would make it more costly to hold tokens for transaction purposes. As this informal argument highlights, it is critical to analyze incentive designs in equilibrium, internalizing the effect of each incentive scheme on the entire network.

More concretely, our primary focus in this chapter will be on incentive mechanisms that encourage participation in block proposal and validation. Participation incentives in cryptocurrencies generally take one of two forms: (1) *transaction fees*, and (2) *validation/block rewards*. Transaction fees are a small fee paid to validators and/or block proposers as a reward for processing a given transaction. In Bitcoin, transaction fees are given to the miner whose block contains the transaction, and are paid by the user who generates the payment. Block rewards, on the other hand, are freshly-minted tokens that are awarded to the node(s) that validate and/or propose a particular block. In some blockchain systems, the roles of block proposers and validators can be separate; for instance, Ethereum proposes such a structure in its future roadmap [183], and Prism naturally separates the functionalities (Chapter 3). In aggregate, transaction fees and block rewards must be valuable enough to incentivize validators to participate in the network. At the same time, transaction fees must be low enough to incentivize participation from transactors. The idea of cryptocurrencies as payment systems has garnered mainstream support based on the premise that transaction fees can be lower relative to traditional services (e.g., credit cards). Thus, at the outset, there exists potential surplus that consumers and network participants aim to realize by reducing transaction frictions. Network participants, like validators, offer services that consume resources and need to be compensated: these services are ultimately compensated with fees paid by consumers. A network design (like Bitcoin) that relies on congestion to induce consumers to offer sufficiently high fees to support network activities is inefficient: while congestion serves as a part of the revelation mechanism, it imposes dead-weight losses on the entire economic system. From this perspective, a design like the PoS system, which instead relies on financial incentives (which are transfers between various actors) is appealing due to its efficiency: it better preserves the total consumer surplus.

7.1 Valuation

Quantifying the valuation of a PoS cryptocurrency is critical, since the security of PoS systems depends primarily on the value of the underlying tokens. Valuation hinges on two key properties of this system. The first property has to do with the blockchain formation mechanism in a PoS

network. Abstracting from the precise details of block validation and block proposal mechanisms, a key property of PoS networks is that the validation protocol requires participating nodes to commit (stake) a certain amount of tokens as collateral. In practice, PoS blockchain protocols typically ask nodes to commit stake for two main purposes: block proposal and block validation. Validation can be thought of either as a periodic finalization process (e.g., the Casper protocol from Ethereum [37]), or more generally, the process of nodes checking the validity of blocks and certifying their correctness in the blockchain. Rewards that accrue to the nodes participating in block proposal and validation are distributed in relation to the size of their committed stake. This protocol, in combination with additional measures (such as penalties for non-participation) is designed to induce proper participation in the validation process.

For our purposes, the main economic implication of this mechanism is that network participants who serve as block proposers or validators can earn rewards by committing their tokens to such activities. In our model, we do not distinguish between the two functionalities; we refer to both simply as 'validation'. Thus, as long as tokens are used for validation, each token generates a stream of cash flows for its holder in the form of additional tokens. This simplified model does not necessarily represent more complex consensus structures (e.g., Prism); however, we use it as a starting point for more nuanced models of PoS consensus.

The second property of the PoS network is that tokens can be re-allocated across various uses: the same token can be used for validation, for consumer and merchant transactions, for off-chain routing of payments, etc. Our analysis assumes, as an approximation, that tokens can be re-allocated in a frictionless manner. If a specific design deviates from this idealized frictionless model, e.g., due to constraints imposed on movements of tokens within the system, one would need to enrich the model with an explicit description of such frictions.

In a frictionless system, individual optimization by the network participants implies that expected rates of return on alternative feasible uses of the tokens must be equalized. In particular, any holder of a token may exchange it for fiat currency and invest the proceeds in a portfolio of financial assets with a return risk profile similar to that of the tokens. Thus, in a system where tokens can be easily exchanged without frictions for fiat currency, the rate of return on holding tokens must equal the opportunity cost: the expected rate of return on the risk-matched investment strategy in financial markets.

7.1.1 A Simple Model with Fee-Based Rewards

We next outline a parsimonious model in which rewards for validation activity are paid entirely in transaction fees generated by retail transactions rather than in newly minted tokens [103]. This model abstracts away from many empirically-relevant details to focus on the main mechanism of token valuation.

Consider a payment network. Suppose that in each period, the network processes retail transactions in the amount of Y_t, measured in the fiat currency, "dollars." Assume that the aggregate float of tokens is fixed, and without loss of generality equal to one. Let p_t denote the equilibrium price of the tokens in terms of the fiat currency. Assume that the network is in a stationary growth regime: the volume of transactions is expected to grow at a constant rate g_Y, so that $\mathbb{E}_t[Y_{t+s}] = Y_t(1+g_Y)^s$.

In this model we do not distinguish between block proposal and block validation functions, and refer to block proposers and block validators simply as "validators." We assume that the protocol for transactions is such that, whenever transactions take place, some tokens must be transferred to the validators as rewards for their activities, and, importantly, individual validators collect their pro-rata share of the total rewards, in proportion to their token stake. The latter assumption rules out individual strategies like selfish mining, which may distort the allocation of rewards among

validators in relation to their individual token balances. We assume that the aggregate amount of fees generated by the network grows at the same long-run rate as the aggregate transaction volume (the ratio of the two series is a stationary process). To simplify the derivations in our model, we further restrict aggregate fees to be a constant multiple of the transaction volume – thus, validators receive fees in the collective amount of cY_t dollars per period (actual rewards are in tokens, which validators sell to consumers in exchange for dollars). We should note that this assumption applies to the total volume of fees, rather than the fee structure for individual transactions. Our assumption of aggregate fee dynamics is consistent with multiple alternative fee schedules, and does not mean that the cryptocurrency must necessarily adopt transaction fees that are a fixed fraction of transaction value; we are still evaluating different design choices. In particular, we discuss the tradeoffs between user-determined transaction fees and algorithmic transaction fees in Section 7.3 below.

We assume that validators have unrestricted access to financial markets and behave competitively: they take market prices, and, importantly, *the design of the payment network*, as exogenous and not affected by their individual actions. Also, we assume that the risk premium associated with a financial claim paying Y_t dollars per period is constant and equal to λ_Y (in equilibrium, this determines the opportunity cost of capital associated with validation activity). Finally, we also assume that there are no physical costs associated with block validation activities.

We look for a stationary equilibrium, in which $\phi \in [0, 1]$ tokens are held by the validators, and $1 - \phi$ are held by consumers for transaction purposes. We assume that validators have no use for tokens outside of their validation activity, and therefore stake their entire token balance. The equilibrium distribution of token holdings, ϕ, is endogenous and determined as a part of the solution.

In equilibrium, the total market value of all the tokens held by the validators is ϕp_t, which is the value of a financial claim on the perpetual stream of cash flows in the amount of cY_t per period. Assuming no valuation bubbles, the market value of this cash flow stream is given by the valuation formula for a perpetuity with constant growth:

$$p_t\phi = \lim_{T\to\infty}\left[\sum_{s=1}^{T}\frac{cY_t(1+g_Y)^s}{(1+\lambda_Y)^s}\right] = \frac{cY_t}{\lambda_Y - g_Y}. \tag{7.1}$$

To pin down the value of the tokens, we need to make an assumption about consumer's demand for tokens. In a market with infinite token velocity, consumers would hold no balances, which would imply that in equilibrium $\phi = 1$ and $p_t = (cY_t)/(\lambda_Y - g_Y)$. More generally, if consumers hold balances equal to k times the transaction volume per period, then $p_t(1 - \phi_t) = kY_t$, and therefore the equilibrium token value is

$$p_t = \left(k + \frac{c}{\lambda_Y - g_Y}\right) Y_t. \tag{7.2}$$

Note that in the absence of an explicit description of demand for token balances, $(cY_t)/(\lambda_Y - g_Y)$ serves as a lower bound on the token value.

The equilibrium value of the tokens in (7.2) consists of two terms. The first term, kY_t, reflects consumer demand for holding token balances. The value of this term depends on k, which is inversely related to the equilibrium token velocity. It is important to acknowledge that token velocity is an equilibrium outcome, related to a number of properties of the payment network and the broader market. Wider adoption of the tokens could raise demand for token balances, while efficient channels for transactions between tokens and fiat currency would enable consumers to support the desired volume of transactions with lower token balances and result in higher token velocity.

The relation between token value and token velocity is commonly invoked when discussing the valuation of cryptocurrencies. While token velocity and equilibrium token value are certainly related

in equilibrium, the relation between the two, like its analog in traditional monetary economics, is not a true structural relation and it does not provide a reliable anchor for token valuation.

The second term in the valuation equation (7.2), $(cY_t)/(\lambda_Y - g_Y)$, reflects the demand for tokens from validators. This term is proportional to the overall volume of retail transactions, Y_t, and to the rate at which fees are charged for transactions. All else equal, broader adoption and utilization of the payment system (higher Y_t) results in higher value of the tokens. Importantly, the above equation does not suggest that token value is increasing in the level of fees. Our analysis here focuses on a single stationary equilibrium and does not explicitly describe how the systems responds to changes in parameters: higher fee levels would eventually lead to lower transaction volume.[1]

Our analysis in this section also relies critically on the assumption of competitive behavior by the network participants. To what extent this assumption offers a good approximation of agent behavior in this environment depends on individual opportunities and incentives to engage in strategic behavior. Ultimately, individual incentives and token valuation are closely linked in PoS systems, and must be analyzed jointly. Such analysis is beyond the scope of this chapter.

7.1.2 An Extended Model with Increasing Token Supply

Here we extend the valuation model of the previous section to allow for increasing token supply. In this model, validators are rewarded in newly minted tokens in addition to the fees collected from consumer transactions. For tractability, we formulate the extended model in continuous time.

We now allow the transaction volume growth parameter to vary in time, and denote it by $g_Y(t)$. We also introduce randomness into the evolution of the transaction volume Y_t, as shown in the following differential equation:

$$\frac{dY_t}{Y_t} = g_Y(t)\,dt + \sigma(t)\,dZ_t,$$

where $g_Y(t)$ and $\sigma(t)$ are bounded continuous functions of time, and Z_t is a Brownian motion.[2] Here Z_t is a simple model for the randomness in the system, and $\sigma(t)$ represents the instantaneous volatility of growth in transaction volume. Investors require compensation for being exposed to dZ_t shocks based on the comparable investment opportunities in financial markets, which we assume to be η units of expected excess returns per unit of risk. η is known as the market price of risk, and we take it to be constant here, for simplicity. Then, under the risk-neutral valuation measure Q, the transaction volume follows

$$\frac{dY_t}{Y_t} = g_Y^Q(t)\,dt + \sigma(t)\,dZ_t^Q,$$

where

$$g_Y^Q(t) = g_Y(t) - \eta\sigma(t),$$

and Z_t^Q is a Brownian motion under measure Q.

As above, p_t denotes the total value of tokens at time t and validators hold fraction ϕ_t of all tokens. The market clearing condition requires that validators and consumers collectively hold all tokens, and thus

$$\phi_t p_t = p_t - kY_t,$$

where k is a consumer demand parameter, which we again assume to be constant. Thus,

$$\phi_t = 1 - k\frac{Y_t}{p_t}.$$

[1] The effect of transaction fees on token value is analogous to the concept of the Laffer Curve in the theory of taxation.

[2] We implicitly assume that all elements of the model, e.g., the growth rate of Y_t, are properly restricted so that tokens have a finite value in equilibrium.

We assume that token supply grows deterministically over time. Specifically, we assume that new tokens are issued at (bounded) rate $r(t)$. In expectation, under the risk-neutral valuation measure, validators earn the risk-free rate of return, which we assume to be constant and denote it by λ_f. Thus, we obtain the valuation equation:

$$\phi_t p_t = cY_t\,dt + e^{-\lambda_f dt} E_t^Q\big[(\phi_t + (1-\phi_t)r(t)\,dt)p_{t+dt}\big].$$

On the left-hand side, $\phi_t p_t$ is the total market value of the tokens staked by the validators at time t. On the right-hand side, we have two terms. The first term, $cY_t\,dt$, is the flow of transaction fees that accrue to the validators over the infinitesimal period $[t, t+dt)$. The second term, $e^{-\lambda_f dt}E_t^Q\big[(\phi_t + (1-\phi_t)r(t)\,dt)p_{t+dt}\big]$, is the discounted expected value (under measure Q)) of the validators' token holdings at the end of the period.[3] Then,

$$E_t^Q\big[dp_t\big] = \lambda_f p_t\,dt - \frac{p_t}{p_t - kY_t}\left(cY_t + kY_t r(t)\right)\,dt.$$

We look for an equilibrium token price process p_t of the form $p_t = p(t, Y_t)$, where $p(t,Y)$ is a sufficiently smooth function of its arguments. Applying Ito's lemma, we obtain a PDE on the token price function:

$$\frac{\partial p}{\partial t} + \frac{\partial p}{\partial Y} g_Y^Q(t) Y + \frac{1}{2}\frac{\partial^2 p}{\partial Y^2}\sigma(t)^2 Y^2 - \lambda_f p + \frac{p}{p - kY}(c + kr(t))Y = 0. \tag{7.3}$$

Equation (7.3) is the valuation PDE. This equation has multiple solutions, and we look for a non-negative solution without valuation bubbles. Specifically, we look for a solution with suitably bounded growth in Y, and subject to a boundary condition

$$p(t,0) = 0,$$

which requires that token value vanishes at zero transaction volume (recall that zero is an absorbing boundary for the transaction volume process). The above equation has a linear solution,

$$p(t,Y) = A(t)Y, \tag{7.4}$$

where the unknown function $A(t)$ is a bounded, positive solution of the ODE:

$$\frac{dA(t)}{dt} + g_Y^Q(t)A(t) - \lambda_f A(t) + \frac{A(t)}{A(t) - k}(c + kr(t)) = 0.$$

In the stationary case of $r(t) = r$, $\sigma(t) = \sigma$, and $g_Y(t) = g_Y$, the total value of tokens p_t is a constant multiple of the transaction volume Y_t:

$$\frac{p_t}{Y_t} = A(t) = k + \frac{c + kr}{\lambda_f + \eta\sigma - g_Y}. \tag{7.5}$$

This solution describes the total value of tokens under constant growth rate in the number of outstanding tokens. Note that, $\lambda_f + \eta\sigma = \lambda_Y$, which is the expected return on the financial claim

[3] To see how this term is determined, note that all new tokens accrue to the validators. Therefore, if validators start the period $[t, t+dt)$ with a fraction ϕ_t of the tokens, they end the period with

$$\frac{\phi_t + r(t)\,dt}{1 + r(t)\,dt} = (\phi_t + r(t)\,dt)(1 - r(t)\,dt) + o(dt) = \phi_t + (1-\phi_t)r(t)\,dt + o(dt)$$

tokens.

paying a cash flow stream equal to the aggregate flow of transaction fees, cY_t. We thus recover the valuation formula (7.2) by setting the token rewards to zero, $r = 0$.

The above solution highlights the valuation effect of rewarding validators in newly minted tokens. In addition to collecting transaction fees, validators also collect proceeds from seignorage. Comparing with (7.2), we see that this effectively raises the flow of proceeds to validators from c to $c + kr$. The second term, kr is intuitive: transfers to validators due to seignorage are proportional to the level of token balances held by consumers. If consumers hold no token balances between transactions, any benefit validators derive from collecting rewards in newly minted tokens is completely offset by the decline in the market value of tokens in their stake.

7.2 Block Rewards

Choosing a block reward schedule is an important question that is often addressed heuristically. For instance, in (7.5), we considered a stationary setting where the rate of growth of the token pool $r(t)$ was constant. In general, we do not expect the system to be stationary; in fact, choosing a block reward function $r(t)$ is of central interest. In this section, we discuss some of the key economic implications of block rewards for incentives of network participants, as well as distributional implications of alternative reward schedules.

A key property of block rewards is that they add to the total token supply. Some cryptocurrencies (e.g., Bitcoin [61] and derivatives) offer a block reward that decreases over time according to a fixed schedule, with a pre-determined end date. The total supply of Bitcoin tokens is thus limited. Others (e.g., Monero) maintain the block reward rate in excess of a positive lower bound at all time [177]; hence the token supply is unbounded. A natural question is how to evaluate these two operating points. In this section, we discuss some of the design tradeoffs.

7.2.1 The economic implications of block rewards

Who pays for running the network?

Growth in total token supply imposes a cost on all token balances, since purchasing power of tokens is eroded over time. This effect is analogous to the well-known concept of "inflation tax" in conventional economic systems. Validators (recall that we group validators together with block proposers) are the beneficiaries of this "tax," since they collect the newly minted tokens in the form of block rewards. As far as valuation goes, block rewards effectively raise the fees collected by the validators. This can be seen from the steady-state valuation equation (7.5): holding all else equal, raising kr, where r is the rate of flow of block rewards, is equivalent to raising the transaction fee parameter c. The precise breakdown between transaction fees and token-based rewards matters, however, because it determines who exactly pays for network operations.

Both types of rewards, transaction fees and block rewards, ensure that all those enjoying the benefits of the network contribute resources to running it. This arrangement closely resembles the funding of commonly used infrastructure in conventional economic systems: for instance, a highway network may be supported through toll fees, paid directly by the drivers, and through public funds, contributed by the broad base of tax payers enjoying the indirect benefits of the highway system. In the context of the electronic payment system, transaction fees are paid by the consumers whenever they use tokens for retail transactions, and accrue to the validators who support operations of the payment network that enables such transactions. In turn, block rewards expressed in newly minted tokens are effectively paid for by *all* network participants who hold token balances, whether or not

they use their tokens for transactions. Importantly, all those who hold tokens as a store of value or for speculative purposes end up bearing some of the cost of such block rewards.

Spreading the burden of funding network operations across a broad range of users, from frequent transactors to long-term holders of tokens, is more than an issue of fairness. All else equal, making speculators pay for some of the block rewards reduces the required level of transaction fees, which generally enhances token adoption and growth of the network. Is is reasonable to expect that, in a new system, the volume of retail transactions starts at a relatively low level, grows over time. In contrast, speculative trading may be very active from the start, when future prospects of the network are highly uncertain, declining eventually as the network settles into its steady-state dynamics. Under such expectations of the network's growth, distributional considerations dictate that one should initiate the network with a relatively high initial block reward rate, so resources needed to support network function at this early stage are contributed primarily by speculators, and gradually reduce block rewards as the volume of retail transactions rises. In the long run, funding of network functions would then increasingly rely on the transaction fees, rather than block rewards.

Incentives for Validators

Consider the steady-state solution for token valuation, (7.4,7.5). Validators earn an expected rate of return equal to $\lambda_f + \eta\sigma$, which is their opportunity cost of taking on the risk of the flow of transaction fees and block rewards. Note that the equilibrium expected return of the validators is neither affected by the level of transaction fees nor the block reward rate. Rather, validators' expected rate of return is determined by the opportunity cost of holding tokens. In equilibrium, each validator decides how many tokens to stake. As validators stake more tokens, they collectively drive down the equilibrium rate of return each of them earns; conversely, their rate of return rises as validators stake fewer tokens. As a result, the level of transaction fees c and block reward rate r affect the combined value of the validators' stakes, but not the equilbrium expected rate of return they earn. Specifically, the combined value of the validators' stakes is

$$\phi_t p_t = (A - k)\, Y_t = \frac{c + kr}{\lambda_f + \eta\sigma - g_Y}\, Y_t.$$

All else equal, the value of validators' token holdings increases in the level of transaction fees c and the block reward rate r. The fraction of the overall token pool held by the validators is also increasing in these two parameters.

Rewarding validators with new tokens thus offers them a claim on the proceeds from seigniorage in the PoS system. Those network participants who hold balances and do not act as validators have no such claim – they maintain token balances because of their convenience value or for speculative reasons. This creates a wedge, $\lambda_f + \eta\sigma - g_Y + r$, between expected returns earned by the validators and returns earned by other holders of tokens. Thus, while not affecting the equilibrium expected rate of return of validators, the rate of growth of token supply r lowers the expected return earned by other network participants. A higher rate of block rewards thus discourages network participants from passively holding tokens and not acting as validators. This mechanism becomes particularly meaningful when we account for various additional costs faced by validators, including physical costs of running validator nodes and losses due to stake slashing (triggered by accidental violations, e.g., validator nodes being off-line).

User Adoption

In the stylized model of this chapter, the volume of consumer-merchant transactions using tokens is exogenous, and consumer holdings of tokens are inelastically related to this transaction volume. In

a more realistic description of network dynamics, consumer demand for token balances should be elastic with respect to the expected rate of return earned on such balances, in addition to the liquidity benefits of holding tokens. By penalizing token balances, higher block reward rates reduce consumer demand for token balances, ultimately translating into lower adoption of tokens by consumers and merchants. In other words, by reducing expected returns on passive holdings of tokens, higher block reward rates may eventually hurt user adoption of the cryptocurrency. Such considerations inform our thinking about the qualitative aspects of the desirable block reward schedule. More formal analysis of user adoption requires further research, and is beyond the scope of this chapter.

7.2.2 Block Reward Schedule: Design Considerations

Our analysis of economic implications of block rewards provides some guidance for how to design block reward schedules with desirable properties. A well-chosen block reward schedule should ensure long-term economic viability of the payment network, including sufficient incentives for user adoption and for participation in performing network functions (block proposals and validation), while also enforcing fairness among validators and proposers. We currently envision a design with a reward rate that starts at a relatively high level at the launch of the network, and decays over time as the network matures.

A high initial rate of block rewards is in line with the design of many existing cryptocurrencies. This encourages participation of token holders in the validation pool during the early period of network development, when the volume of consumer transactions, and thus the volume of transaction fees is low. During this early period, it is imperative to reward proposer and validator nodes for their efforts; without such nodes, the system cannot operate.

As adoption increases, initial growth in token supply will gradually decline. In the long run, the desirable rate of block rewards is subject to the tradeoffs discussed in Section 7.2.1 above. Higher reward rate distributes the financial burden of running the network over a broader base of token holders. At the same time, higher reward rate effectively taxes passive token balances, hurting user adoption. Depending on which of these forces dominates, the long-run block reward rate may remain positive or possibly converge to zero. We are still in the process of selecting a block reward schedule.

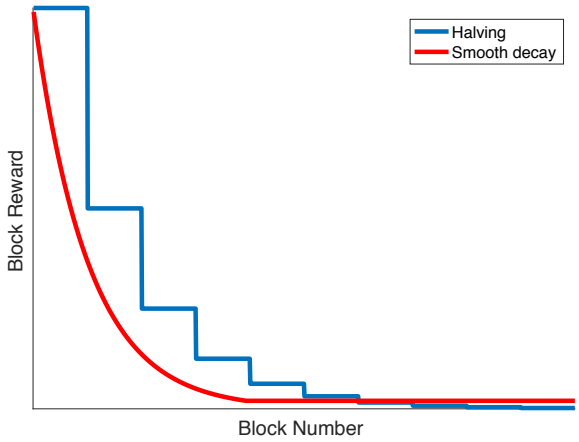


Figure 7.1: Comparison between common block reward functions. Halving is used in Bitcoin and many of its derivatives, whereas smooth decay is used in Monero.

For reference, Figure 7.1 shows caricatures of reward rate schedules that have been adopted in

existing cryptocurrencies. 'Halving' refers to reward schedules that maintain a constant reward over a long period (e.g. years), and periodically reduce the block reward by half. This approach is used in Bitcoin and several cryptocurrencies derived from it. 'Smooth decay' is used in Monero, and gives a smooth approximation of the reward function in Bitcoin. Notice that the smooth decay curve is lower bounded by a fixed minimum block reward; this is a detail specified by Monero's monetary policy. In our case, that minimum rate of growth in token supply need not exist.

We show in the next section (7.2.3) that choosing large initial block rewards (compared to the initial stake pool) can lead to substantial and irreversible wealth imbalance in PoS systems. For example, Bitcoin started with a stake pool of 50 BTC and initial block rewards of 50 BTC per block; in a PoS system, such parameter choices could very easily lead to substantial wealth concentration. Starting with a large initial stake pool compared to the block reward size is an easy way to counter these effects. At first glance, this directive seems to counter the earlier conclusion that we need high initial inflation to encourage adoption. Indeed, there is a tradeoff between choosing inflation high enough to encourage adoption and keeping it low enough to prevent severe wealth imbalance. To this point, notice that to prevent stake imbalance, the inflation rate does not need to be excessively small. In practice, our simulations suggest that if the block reward is less than, say, a thousandth of the initial stake pool, then stake imbalance remains mild [68]. For example, if Unit-e begins with an initial stake pool of 1.5 billion tokens, then this rule of thumb suggests a block reward of at most 1.5 million tokens per block. Aggregated over all the blocks in a year at a (conservative) block rate of 30 seconds per block, this imposes an upper bound on annual inflation of around 100,000%. Such an inflation rate is orders of magnitude higher than any we would set in practice, even in the early days of Unit-e. Hence, the apparent tension between encouraging equitability and adoption is weak; for a large enough stake pool, we can choose inflation to be very high without affecting equitability.

7.2.3 Equitability in PoS block reward schemes

In this section, we study one of the tradeoffs involved in choosing a block reward function: equitability. At a high level, equitability means that a party that begins with a certain fraction of stake cannot unduly amplify its fraction of stake simply by participating in block proposal or validation. Perhaps surprisingly, standard PoS implementations can exhibit extreme inequitability if system parameters are chosen naively. For example, consider a PoS blockchain that awards the full block reward to the user that proposes each block. This is a standard assumption that is used in almost every PoS cryptocurrency today. Suppose further that block rewards are immediately incorporated into each party's portion of the proposal stake pool. Now let us initialize such a system with parameters comparable to Bitcoin's initial state: 50 initial tokens and a block reward of 50 tokens. If party A owns 1/3 of the initial stake, a simple Polya's urn argument shows that as the block length tends to infinity, the limiting stake of party A converges almost surely to a random variable with distribution Beta($\frac{1}{3}, \frac{2}{3}$), visualized in Figure 7.2 by the blue solid curve labelled 'Constant Rewards, PoS'.

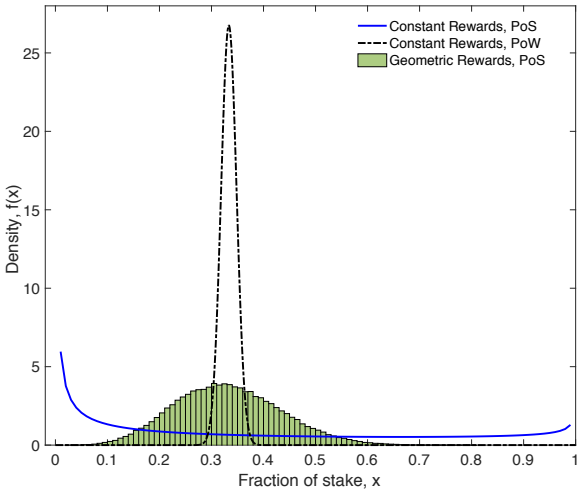


Figure 7.2: Limiting stake distribution of a party with 1/3 stake, under a PoS and PoW system initialized with Bitcoin's block reward parameters.

An important and troublesome property of this Beta distribution is the high probability mass near 0 and 1. Effectively, this means that party A can end up with extremely high or low stake percentages, simply through the natural randomness of the PoS leader election protocol. Moreover, this convergence occurs quickly in practice. In contrast, the black dashed curve illustrates the stake distribution for PoW after $T = 1,000$ blocks, using the same constant block reward function as Bitcoin. Notice that the PoW curve is substantially more concentrated around party A's initial stake deposit. A natural question is whether any block reward function can produce a limiting stake distribution comparable to the black curve in a PoS setting. Our objective in this section is to explore this phenomenon more formally; we theoretically demonstrate how compounding affects fairness in PoS systems, both as an effect of the block reward function and the initial conditions of the system. To this end, we introduce a new metric called *equitability*, which intuitively measures how much a party can augment or decrease their fractional stake over time. We show that *no* block reward function can achieve the same degree of equitability as PoW, while also introducing a new *geometric* reward function that is maximally equitable over all reward functions. Figure 7.2 illustrates the empirical stake distribution of geometric rewards over $T = 1,000$ slots as a green histogram; notice that it is closer, but not the same, as the PoW baseline. This is a fundamental price we pay for the efficiency of PoS. Despite the fact that PoS cannot achieve the same equitability as PoW, we find that through proper system initialization, these effects can be dramatically mitigated. In particular, our findings give concrete guidelines for choosing the size of the initial stake pool, and the amount of rewards that should be disseminated over time, in order to ensure an acceptable level of equitability.

Model We begin with some notation modeling a simple chain-based proof-of-stake system. We consider m parties: $\mathcal{A} = \{A_1, \ldots, A_m\}$. We assume that all parties keep all of their stake in the *proposal stake pool*, which is a pool of tokens that is used to choose the next proposer. As before, we consider a discrete-time system, $n = 1, 2, \ldots, T$, where each time slot corresponds to the addition of one block to the blockchain. In reality, new blocks may not arrive at perfectly-synchronized time intervals, but we index the system by block arrivals. For any integer x, we use the notation $[x] := \{1, 2, \ldots, x\}$. We call the time interval $[T]$ an *epoch*: a period of time (blocks) during which a pre-specified number of inflation tokens should be dispensed. In Bitcoin, for example, an epoch lasts

about four years, or 210,000 blocks; this is the time period over which the block reward function remains constant prior to the halving events.

For all $i \in [m]$, let $S_{A_i}(n)$ denote the total stake held by party A_i in the proposal stake pool at time n. We let $S(n) = \sum_{i=1}^{m} S_{A_i}(n)$ denote the total stake in the proposer stake pool at time n, and $v_{A_i}(n)$ denotes the *fractional stake* of node A_i at time n: $v_{A_i}(n) = \frac{S_{A_i}(n)}{S(n)}$. For simplicity, we normalize the initial stake pool size to $S(0) = 1$; each party starts with $S_{A_i}(0) = v_{A_i}(0)$ fraction of the original stake.

At each time slot $n \in [T]$, the system chooses a proposer node $W(n) \in \mathcal{A}$ such that

$$W(n) = \begin{cases} A_1 & w.p. \quad v_{A_1}(n) \\ \dots & \\ A_m & w.p. \quad v_{A_m}(n). \end{cases} \tag{7.6}$$

Upon being selected as a proposer, $W(n)$ appends a block to the blockchain. As compensation for this service, $W(n)$ receives a block reward of $r(n)$ stake, which is immediately added to its allocation in the proposer pool. That is,

$$S_{W(n)}(n+1) = S_{W(n)}(n) + r(n).$$

The reward $r(n)$ is freshly-minted at each time step, so it causes the total number of tokens to grow. We assume the total reward dispensed in time period T is fixed, so $\sum_{n=1}^{T} r(n) = R$.

Consider a *closed* PoS system where no stake is taken out of or put into the proposal stake pool over a fixed period of time T. The only changes to the stake of each party during this interval are due to the compounding effects of the rewards earned in proposing new blocks. At each (discrete) time step, a block reward is dispensed according to the randomized proposer election in Eq. (7.6). This is immediately added to the stake of the elected proposer. We focus on the scenario where every party follows protocol as a starting point.

Common reward functions Between checkpoints, we can choose any block reward schedule $r(n)$ that respects the total reward constraint of R. A natural choice, famously adopted in Bitcoin, is the *constant block reward function*, defined as:

$$r_c(n) := \frac{R}{T}. \tag{7.7}$$

Perhaps surprisingly, we find that r_c introduce inequitability to the block reward scheme if initialized improperly. Critically, this can happen *without any strategic behavior* from any of the participants. Rather, it is the combined effect of compounding stake and randomized leader election that causes this effect. A main result of this work is that a *geometric reward function*, defined as

$$r_g(n) := (1+R)^{\frac{n}{T}} - (1+R)^{\frac{n-1}{T}}, \tag{7.8}$$

provably minimizes inequitability and exhibits a number of other desirable properties. Although geometric rewards may not be appropriate for deployment in a real cryptocurrency for various practical reasons, this work helps us to methodically choose system parameters, such as initial stake pool size.

7.2.3.1 Equitability

We study the stochastic dynamics of the fractional stake of one of the participating parties A, who started with $v_A(0)$ fraction of the initial total stake of $S(0) = 1$. This is without loss of generality, as the random process is homogeneous in scaling both R and $S(0)$ by a constant. We denote this fractional stake at time n by $v_{A,r}(n)$, to make the dependency on the reward function explicit.

One desired property of a closed and fair PoS system is that the fractional stake should remain constant over time in *expectation*. If A contributes 10% of the proposal stake pool at the beginning of the time, then A should, on average, reap 10% of the total disseminated rewards. As randomness in the proposer election is essential to current PoS systems, this cannot be ensured deterministically. Instead, the popular choice of electing a proposer (in Eq (7.6)) with probability proportional to the fractional stake ensures that this holds in expectation. Formally, for all $n \in [T]$,

$$\mathbb{E}[v_{A,r}(n)] \;=\; v_A(0)\;. \tag{7.9}$$

The expected fractional reward for any party stays constant in expectation, irrespective of which reward function is used. This follows from the law of total expectation and the fact that

$$\begin{aligned}
&\mathbb{E}[v_{A,r}(n) \,|\, v_{A,r}(n-1) = v] \\
&\quad = v\frac{v\,S(n-1) + r(n-1)}{S(n)} + \big(1-v\big)\frac{v\,S(n-1)}{S(n)} \\
&\quad = v\;,
\end{aligned}$$

where $S(n)$ is the total stake at time n.

Even with the same expected fractional stake, the choice of the reward function can change the distribution of the final stake dramatically. We propose using the variance of the final fractional stake, $\mathrm{Var}(v_{A,r}(T))$, to measure the level of uncertainty for different choices of the reward function.

Definition 7.2.1. *For two reward functions $r_1 : [T] \to \mathbb{R}^+$ and $r_2 : [T] \to \mathbb{R}^+$ with the same total reward, $\sum_{n=1}^{T} r_1(n) = \sum_{n=1}^{T} r_2(n)$, we say r_1 is more* equitable *than r_2 if*

$$\mathrm{Var}\big(v_{A,r_1}(T)\big) \;\leq\; \mathrm{Var}\big(v_{A,r_2}(T)\big)\;. \tag{7.10}$$

With this notion of equitability, there is a core optimization problem of interest to a PoS system designer: given a fixed total reward R to be dispensed, how do we distribute it over the time T to achieve the highest equitability? Perhaps surprisingly, we show that this optimization has a simple closed form solution.

Theorem 2. *The geometric reward r_g defined in* (7.8) *achieves the highest equitability.*

If equitability is a primary design goal, this gives the optimal reward function. Intuitively, one might prefer to dispense larger rewards towards the end, when the stake pool is also large. At the same time, one needs to dispense some reward in the beginning to grow the stake pool. The optimal trade-off is met with the geometrically increasing reward.

Proof of Theorem 2. We prove that

$$\mathrm{Var}\big(v_{A,r_g}(T)\big) \;\leq\; \mathrm{Var}\big(v_{A,r}(T)\big)\;, \tag{7.11}$$

for all $r \in \mathbb{R}^T$ such that $\sum_{n=1}^T r(n) = R$ and $r(n) \geq 0$ for all $n \in [T]$. To this end, we prove that r_g is a unique optimal solution to the following optimization problem:

$$\begin{aligned} \text{minimize}_{r \in \mathbb{R}^T} \quad & \text{Var}(v_{A,r}(T)) \\ \text{s.t.} \quad & \sum_{n \in [T]} r(n) = R \\ & r(n) \geq 0 \ , \ \forall n \in [T] \ , \end{aligned} \tag{7.12}$$

From the analysis of a time-dependent Pólya's urn model [141], we know the variance from the following formula.

Lemma 7.2.2. *Let $e^{\theta_n} \triangleq S(n)/S(n-1)$, then*

$$\text{Var}(v_{A,r}(T)^2) = \big(v_A(0) - v_A(0)^2\big)\Big(1 - \frac{S(0)^2}{S(T)^2} \prod_{n=1}^T (2e^{\theta_n} - 1)\Big) \ . \tag{7.13}$$

After some affine transformation and taking the logarithmic function of the objective, we get an equivalent optimization of

$$\begin{aligned} \text{maximize}_{\theta \in \mathbb{R}^T} \quad & \sum_{n=1}^T \log(2e^{\theta_n} - 1) \\ \text{s.t.} \quad & \sum_{n \in [T]} \theta_n = \log(1+R) \ , \end{aligned} \tag{7.14}$$

$$\theta_n \geq 0, \forall n \in [T] \ , \tag{7.15}$$

This is a concave maximization on a (rescaled) simplex. Writing out the KKT conditions with KKT multipliers λ and $\{\lambda_n\}_{n=1}^T$, we get

$$\frac{2e^{\theta_n}}{2e^{\theta_n} - 1} - \lambda_n - \lambda = 0 \tag{7.16}$$

$$\lambda_n \geq 0 \tag{7.17}$$

$$\theta_n \lambda_n = 0 \tag{7.18}$$

A solution θ to these conditions must satisfy that θ_n is either zero or $(1/\|\theta\|_0)\log(1+R)$. Among these solutions, we show that $\theta^* = ((\log(1+R))/T)\,\mathbb{1}$ is the unique optimal solution, where $\mathbb{1}$ is a vector of all ones. Consider a solution of the KKT conditions that is not θ^*. Then, we can strictly improve the objective by the following operation. Let $i, j \in [T]$ denote two coordinates such that $\theta_i = 0$ and $\theta_j \neq 0$. Then, we can create $\tilde{\theta}$ by mixing θ_i and θ_j, such that $\tilde{\theta}_n = \theta_n$ for all $n \neq i, j$ and $\tilde{\theta}_i = \tilde{\theta}_j = (1/2)\theta_j$. We claim that $\tilde{\theta}$ achieves a smaller objective function as $\log(2e^{\theta_j} - 1) < 2\log(2e^{\theta_j/2} - 1)$. This follows from Jensen's inequality and strict concavity of the objective function. Hence, θ^* is the only fixed point of the KKT conditions that cannot be improved upon.

In terms of the reward function, this translates into $S(n)/S(n-1) = (1+R)^{1/T}$ and $r(n) = (1+R)^{n/T} - (1+R)^{(n-1)/T}$.

□

7.2.3.2 Equitable choices of the total reward R

The equitability of a system is determined by four factors: the number of block proposals T, choice of reward function r, initial stake of a party $v_A(0)$, and the total reward R. Note that because we normalized the initial stake $S(0)$ to be one, R represents the ratio between the total rewards dispensed and the initial stake $S(0)$. Some of these factors should be chosen with respect to external factors (e.g. interest rate and valuation of the crypto currency) and practical considerations (e.g. incentivizing participation of proposers). In this section, we study the scenario where the system designer has the freedom to choose R, either by setting the initial stake size $S(0)$ and/or by setting the total reward during T. The focus in how equitability trades off with the choice of R, and how this trade-off changes with respect to the choice of the reward function r during this interval.

Concretely, we consider a scenario where T is a large enough integer, and $v_A(0)$ of each party is also fixed as it is not up to the system designer to choose. To make the system more equitable, we want to minimize the variance of the final stake of all parties simultaneously. We study what choices of R and r achieve a desired level of variance, normalized by $(v_A(0) - v_A(0)^2)$ which is the worst-case variance for each party. This happens when the fractional stake $v_A(T)$ converges to a binary distribution with support set $\{0, 1\}$. As $\mathbb{E}[v_A(T)] = v_A(0)$, the variance of this Bernoulli distribution is $v_A(0) - v_A(0)^2$. Among all distributions supported on $[0, 1]$ with mean $v_A(0)$, this has the largest variance.

Geometric reward function. For $r_g(n)$, we have $e^{\theta_n} = (1 + R)^{1/T}$. It follows from Lemma 7.2.2 that

$$\frac{\mathrm{Var}(v_{A,r_g}(T))}{v_A(0) - v_A(0)^2} = 1 - \frac{(2(1+R)^{1/T} - 1)^T}{(1+R)^2}\,, \tag{7.19}$$

When R is fixed and we increase T, we can distribute small amounts of rewards across T and achieve vanishing variance. On the other hand, if R increases much faster than T, then we are giving out increasing amounts of rewards per time slot and the uncertainty grows. This follows from the above variance formula, which we make precise in the following.

Proposition 7.2.3. *For a closed PoS system with a total reward $R(T)$ chosen as a function of T and a geometric reward function $r_g(n) = (1 + R(T))^{n/T} - (1 + R(T))^{(n-1)/T}$, it is sufficient and necessary to set*

$$R(T) = \left(\Big(\frac{1}{1 - \sqrt{\frac{\log(1/(1-\varepsilon))}{T}}} \Big)^T - 1 \right) \big(1 + o(1)\big)\,, \tag{7.20}$$

in order to achieve $\lim_{T\to\infty} Var(v_{A,r_g}(T)) = \varepsilon(v_A(0) - v_A(0)^2)$.

This follows from substituting the choice of $R(T)$ in the variance in Eq. (7.19), which gives

$$\begin{aligned} \lim_{T\to\infty} \frac{\mathrm{Var}(v_{A,r_g}(T))}{v_A(0) - v_A(0)^2} &= \lim_{T\to\infty} 1 - \Big(1 - \frac{\log(1/(1-\varepsilon))}{T}\Big)^T (1 + o(1)) \\ &= \varepsilon\,, \end{aligned} \tag{7.21}$$

This is monotonically non-decreasing in R and non-increasing in T, as expected from our intuition. For example, if R is fixed, one can have the initial stake $S(0)$ as small as $\exp(-\sqrt{T}/(\log T))$ and still achieve a vanishing variance. As the geometric reward function achieves the smallest variance (Theorem 2), the above $R(T)$ is the largest reward that can be dispensed while achieving a desired

normalized variance of ε in time T (with initial stake of one). This scales as $R(T) \simeq (1+1/\sqrt{T})^T \simeq e^{\sqrt{T}}$. Equivalently $R/R(T)$ is the smallest initial stake one can start with, while dispensing R rewards in time T. We need more initial stake or less total reward, if we choose to use other reward functions.

Geometric rewards may not be appropriate in practice. If one were to concatenate multiple epochs with geometric reward functions, the drop in block rewards from the end of one epoch to the beginning of the next could be dramatic. This could cause proposers to (temporarily) leave the system, thereby preventing blockchain growth. Hence for purely practical reasons, geometric rewards may be a less desirable solution than constant rewards. We therefore include analogous variance calculations for the constant reward function. Specifically, our focus is on understanding how large the initial stake pool needs to be in order to ensure low variance in proposer stake.

Constant reward function. Consider the constant reward function of Eq. (7.7). As $e^{\theta_n} = (1+nR/T)/(1+(n-1)R/T)$, it follows from Lemma 7.2.2 that

$$\begin{aligned} \frac{\mathrm{Var}(v_{A,r_c}(T))}{v_A(0)-v_A(0)^2} &= 1 - \frac{1+R+\frac{R}{T}}{1+R+\frac{R}{T}+\frac{R^2}{T}} \\ &= \frac{R^2}{(T+R)(1+R)} \,. \end{aligned} \tag{7.22}$$

Again, this is monotonically non-decreasing in R and non-increasing in T, as expected. The following condition immediately follows from Eq. (7.22).

Proposition 7.2.4. *For a closed PoS system with a total reward $R(T)$ chosen as a function of T and a constant reward function $r_c(n) = R(T)/T$, it is sufficient and necessary to set*

$$R(T) = \frac{\varepsilon\, T}{1-\varepsilon}\,(1+o(1))\,, \tag{7.23}$$

in order to achieve $\lim_{T\to\infty} Var(v_{A,r_c}(T)) = \varepsilon(v_A(0)-v_A(0)^2)$.

By choosing a constant reward function, the cost we pay is in the size of the total reward, which can only scale as $O(T)$. Compared to $R(T) \simeq e^{\sqrt{T}}$ of the geometric reward, there is a significant gap. Similarly, in terms of how small initial stake can be with fixed total reward R, constant reward requires at least $S(0) \simeq R/T$. Hence by choosing $S(0)$ to be at least a constant fraction of the sum block rewards over time, we can ensure low variance even with a constant reward function.

7.3 Transaction Fees

Another class of incentives known as *transaction fees* are drawn from users' transactions and awarded to nodes who process that node's transactions. This can include both the block proposer and/or validator. A primary question in cryptocurrencies is how to choose transaction fees. A common approach—used in Bitcoin and Ethereum, for example—is to allow users to propose their own fees for each transaction. In such a system, miners are incentivized to prioritize high-fee transactions when forming blocks; the higher the transaction fee, the lower the confirmation delay. Most cryptocurrencies also set a minimum transaction fee to prevent users spamming the network with transactions. Two important questions characterize the selection of transaction fees. The first is how to choose the magnitude of transaction fees; namely, should fees be proposed by users, imposed by the system, or some combination thereof? Typical considerations include scaling fees with the

transaction's bytesize, monetary value, or both. The second question is how to allocate fees once they are collected. Recent work has shown that naive (and very common) methods of allocating fees can lead to incentive-incompatibility as well as increased confirmation latency. We begin this section by presenting an industry-standard approach to fee management in Section 7.3.1, and discussing why such an approach is unacceptable for a low-latency payment system. Next, we discuss alternatives, and highlight their advantages and disadvantages 7.3.2.

7.3.1 Fee management today

Most existing cryptocurrencies allow users to set their own fees; validators choose whether to process a given transaction based on whether the offered transaction fee is high enough. If a miner (or proposer) includes a transaction in a block, that node reaps the full transaction fee as a reward. If a given transaction is not included in a block by any miner after some period of time, that same transaction can be re-broadcast with a higher fee. Although Bitcoin and Ethereum place lower bounds on the size of transaction fees, in practice, users tend to choose substantially higher transaction fees. This natural approach to fee selection is widely adopted, but it has three important negative repercussions: high congestion, high fees, and incentive incompatibility. We begin by describing each of these repercussions more precisely.

Congestion Current fee systems contribute critically to congestion. Intuitively, this happens because current mechanisms incentivize validators to choose which transactions to confirm based on transaction fees; this in turn forces users to pay to bypass congestion at equilibrium. To make this statement precise, consider a simplified system in which new blocks are produced as a Poisson process of rate μ, each of which can contain up to K transactions. Transactions arrive in the system at rate λ, and each transaction offers a transaction fee b, chosen by the user. Let c denote the (random) cost per unit time of delaying a transaction's confirmation, with $c \sim F[0, \bar{c}]$, where $\bar{c}$ is a positive constant. So if a single transaction's confirmation is delayed by time Δ, a delay cost of $c\Delta$ is incurred. Here $F(\cdot)$ denotes a cdf with density $f(\cdot)$ and tail probability $\bar{F}(c) := 1 - F(c)$. The congestion in the system ρ is defined as $\rho := \frac{\lambda}{\mu K}$. Let $W_K(\rho)$ denote the expected transaction confirmation delay in a system (in blocks) with block size K and congestion ρ, $R_K(\rho)$ the total revenue per unit time raised from users, and $D_K(\rho)$ the total delay cost incurred per unit time. Consider the following result by Huberman *et al.*:

Theorem 3 ([94]). *At equilibrium, if $\rho = 0$, both revenue and delay cost are 0. For all $\rho \in (0, 1)$,*

$$\begin{aligned} R'_K(\rho) &= K\rho \int_0^{\bar{c}} \bar{F}(c)^2 W'_K(\rho \bar{F}(c)) dc > 0, \\ D'_K(\rho) &= \frac{R_K(\rho) + D_K(\rho)}{\rho} > 0. \end{aligned}$$

In other words, both revenue (and with it infrastructure provision by miners) and delay cost are strictly increasing in ρ.

A primary, troubling repercussion of this result is that congestion is not merely a side effect of technical scalability challenges. Rather, it is fundamental; validators are not incentivized to work without congestion. As stated in [94]:

> An implication of [Theorem 3] is that congestion and delays are necessary for the system to function. Low congestion ρ leads to low delay costs, as blocks are rarely full and each transaction is likely to be processed in the next block. But when blocks are rarely full

> users have little incentive to pay transaction fees to gain priority, and the system raises little revenue. Without sufficient revenue the number of miners... can become too small, making the system unreliable.

This instability among miners under low congestion conditions is discussed more carefully when we consider incentive compatibility. We want to highlight that although [94] was written for the PoW setting, their results are not specific to PoW. Rather, they stem from the combination of three factors: (1) blocks are produced at a constant rate with a limited block size, (2) different transactions give miners (proposers) different reward amounts, and (3) miners get to choose which transactions to include in each block. If a similar fee mechanism were adopted in a PoS system, the same effects would emerge, due to the fact that block times are lower bounded (by protocol and/or by physical constraints). This fundamental relation between congestion and user-selected fees is problematic in a high-throughput payment system.

High fees Another substantial problem with the current fee structure is that fees can be unbounded. Consider the following result from Huberman *et al.*, which considers the same model from the prior discussion on congestion:

Theorem 4 ([94])**.** *At equilibrium, transaction fees coincide with the payments that result from selling priority of service in a Vickrey-Clark-Groves (VCG) auction.*

A well-known property of VCG auctions is that participants are incentivized to bid an amount equal to the externality they impose upon others. This may not be problematic on average, but these amounts can become artificially inflated due to social factors, world events, random fluctuations in system load, etc. Indeed, in December of 2017, Bitcoin fees exceeded $20 per transaction [92]. This volatility and unboundedness is explicitly at odds with the requirements of a payment system. Merchant adoption hinges critically on the guarantee that fees will *not* grow beyond a maximum threshold; this point has been touted as a primary reason why cryptocurrencies can never replace fiat money [164].

A key observation is that whereas congestion arises mainly because of the way fees are disbursed, high or volatile fees result from a combination of two factors: first, fees are proposed by users. Second, validators receive higher rewards for transactions with higher fees, and choose which transactions to include. It is this combination that leads to the VCG result, which in turn leads to unpredictable fees. Hence, any solution to this problem should tackle both factors; it is not enough to simply change the way that fees are disbursed.

Incentive incompatibility A third critical repercussion of the current approach to transaction fee management is incentive incompatibility. In particular, in underutilized systems, it can introduce undesirable equilibria in terms of proposal strategies. Carlsten *et al.* show that in the absence of inflation, rational Bitcoin miners are no longer incentivized to mine on the longest chain [40]. The intuition is that if miners are only collecting transaction fees, then they are incentivized to include as many transactions as possible in each block. This means that right after a block is mined, the number of transactions available for inclusion in the next block is minimal, as is the associated reward. Thus, when choosing between forks, miners are incentivized to adopt one with fewer transactions in the final block, since such forks leave more transaction fees for the next miner. By exploiting this observation, selfish miners can convince other miners to build on side chains.

More precisely, [40] shows the existence of equilibria that do *not* involve honest mining. Their model is as follows: consider a constant stream of incoming transactions. For any time interval $\mathcal{T} = [t_1, t_2]$, let $g(\mathcal{T}) = t_2 - t_1$ denote the fees accumulated by the transactions in interval $\mathcal{T}$ (i.e.

one unit of fees per unit time). Notice that the model considers transactions as continuous quantities that can be infinitesimally split between miners. Because of the one-to-one correspondence between transaction times and associated fees, we will interchangeably allow $\mathcal{T}$ to denote both a time interval and a set of transactions; the meaning should be clear from context. Suppose there is no network latency, and no limit on block size (equivalently, there are few enough transactions in the system that a miner can always include all outstanding transactions in the next block). For a block B, let o_B denote the node that proposed block B. A key assumption is that when a miner mines a block at time t on block B, it can choose to include any subset $\mathcal{T}' \subseteq [0, t]$ of outstanding transactions in the new block, thereby collecting fees $g(\mathcal{T}') = |\mathcal{T}'|$. Let $L_B(t)$ denote the amount of outstanding transactions in interval $[0, t]$, i.e., transactions that have *not* been claimed by B or its predecessors. For an amount of outstanding transactions $L_B(t)$, suppose agents use function $f(L_B(t))$ to determine how many transactions to consume in the next block. The authors also assume that if the chain has height H, a miner will either build on the chain at height $H + 1$ or fork the last block, and build a conflicting block at height H; let H_B denote the height of block B. Let M_H denote the block at height H that leaves the most unclaimed transactions, i.e.,

$$M_H = \arg\max_B \{L_B(t) \,:\, H_B = H\}.$$

Similarly, let $G_H(t) = L_{M_{H-1}}(t) - L_{M_H}(t)$ denote the gap between the most unclaimed fees available at height $H - 1$ and the most unclaimed fees at height H. [40] considers a class of strategies where each miner chooses (a) which block to build on, and (b) how many transactions to consume in their block. Algorithm 1 describes this procedure.

Algorithm 1 Function-fork(f) [40]

Require: Transaction selection function f
  for node m mining at time t **do**
    if $o_{M_H} = m$ or $f(L_{M_H}(t)) > \min\{f(L_{M_{H-1}}(t)), G_H(t)\}$ **then**
      build on block M_H
      include $f(L_{M_H}(t))$
    else
      build on block M_{H-1}
      include $\min\{f(L_{M_{H-1}}(t)), G_H(t)\}$
    end if
  end for

Theorem 5 ([40])**.** *For any constant y such that $2y - \ln(y) \geq 2$, define*

$$f(x) = \begin{cases} x & \text{if } x \leq y \\ -W_0(-ye^{x-2y}) & \text{if } x \in (y, 2y - \ln(y) - 1) \\ 1 & \text{else,} \end{cases} \tag{7.24}$$

where $W_0(\cdot)$ denotes the upper branch of the Lambert W function which satisfies $W_0(xe^x) = x$ for all $x \in [-\frac{1}{e}, \infty)$. Then it is an equilibrium for every miner to mine with strategy Function-fork(f) (Algorithm 1) as long as every miner is non-atomic, and miners only mine on top of chains of length H or $H - 1$, where H denotes the length of the longest chain in the system. Furthermore, in such an equilibrium, the expected number of unconfirmed transactions at time $n \in \mathbb{R}$ scales as $\Omega(\sqrt{n})$.

	Miner gets full transaction fees	**Decoupled rewards**
User-selected fees	Bitcoin [130] Ethereum [186]	Cardano [101, 100] Fruitchains [137]
Algorithmic fees		

Table 7.1: Design space for transaction fee mechanisms. To the best of our knowledge, no blockchain systems today chooses transaction fees purely algorithmically. That being said, several cryptocurrencies impose algorithmic lower bounds on fees, but allow users to offer higher fees if desired (e.g., Bitcoin, Cardano, Ethereum).

This theorem suggests that Bitcoin is not stable when miners are incentivized only with transaction fees, whereas block rewards help stabilize the reward against temporal fluctuations in the transaction pool size. The $\Theta(\sqrt{n})$ transaction backlog is problematic, and the strategic incentives are not just edge cases; the paper shows numerically that honest behavior is itself not a Nash equilibrium. Möser and Böhme observe related behavior in a longitudinal measurement of Bitcoin's transaction history [129].

As with the results on congestion and fee volatility, this result is not specific to PoW. Rather, it results from miners deriving different rewards from different transactions, and hence being incentivized to process high-fee transactions. Broadly, there are two approaches for protecting against the challenges presented in [40]. The first is to ensure that there are always more transactions than there is space in each block: essentially, create congestion. This scenario is prevalent in Bitcoin today, which may help explain why these attacks have not been observed in the wild. However, our goal in adopting PoS is precisely to avoid congestion, especially as a means of network control. A second approach is to decouple block rewards from the transactions they process. We explore such options in the following section.

7.3.2 Design considerations

The transaction fee structure for Unit-e has not been finalized. We are particularly interested in two main design choices: (1) whether fees are chosen algorithmically or selected by users, and (2) how transaction fees are disseminated to users. This design space is summarized in Table 7.1, along with a partial listing of current cryptocurrencies in each operating point. In this section, we will describe each of the quadrants in Table 7.1 along with the corresponding tradeoffs.

7.3.2.1 User-selected fees, miner gets full transaction fees

This operating point is by far the most popular today, being used in Bitcoin, Ethereum, Monero, and others. We discussed the drawbacks of this approach in detail in Section 7.3.1. To summarize, there are three main problems with this approach: (1) it incentivizes miners to create congestion, (2) transaction fees are unbounded, and (3) it is susceptible to strategic behavior. Because of these problems, which have been well-documented both theoretically and in practice, we do not view this operating point as viable for Unit-e.

7.3.2.2 User-selected fees, decoupled rewards

Many of the problems from Section 7.3.1 arise because miners can substantially alter their transaction fee rewards by changing which transactions they include in each block. One way to address this

problem is by *decoupling* the transaction fees in a given block from the reward collected by the block miner. If miners cannot significantly alter their rewards by optimizing the transactions in blocks, they are less incentivized to prioritize certain transactions over others.

This operating point is not widely used in live cryptocurrencies, with the exception of Cardano [100]. Nonetheless, the idea of (partially) decoupling a transaction's fee from the miner of the block containing that transaction has been explored in several works, including inclusive blockchains [113], Fruitchains [137], and Ouroboros [101]. Many of the papers studying this idea are designed for the PoW setting. The basic idea is that the miner of a block does not necessarily reap all the transaction fees in that block, but shares them with other miners who have also contributed to network security and validation. There are many ways to implement such a directive; one explicit mechanism is proposed in Fruitchains, where transaction fees are split evenly over a sequence of consecutive blocks. A key property that Fruitchains exhibit is *fairness*, defined below.

Definition 7.3.1 ([137]). *Consider a Byzantine adversary that controls at most fraction β of compute power (stake); the remaining compute power (stake) is controlled by an honest party. A blockchain protocol is* δ-approximately fair *if there exists a time duration T_0 such that for all $T \geq T_0$, in the view of each honest player, the honest party is guaranteed to obtain at least fraction $(1-\delta)(1-\beta)$ of blocks with overwhelming probability in every time window of duration T.*

Fairness is closely related to our notion of equitability from Section 7.2.3. A key difference is that equitability only makes sense for random processes with fixed mean over time (i.e., processes that are stationary up to order 1), whereas fairness encompasses general non-stationary stochastic processes.

Fruitchains proposes a mechanism whereby instead of directly including transactions in blocks, miners mine a "fruit" for each transaction [137]. When a transaction first appears in the system, miners start solving a proof-of-work puzzle specific to that transaction, with a different hardness parameter from the main blockchain. This proof-of-work puzzle must include a pointer to a recent block within the previous $T(\kappa)$ blocks, where κ is a security parameter, and $T(\kappa)$ is a (pre-specified) polynomial function of the security parameter. A solution to this puzzle, concatenated with the puzzle itself, is called a fruit, and gets broadcast to the network. Regular block miners include fruits in their blocks, rather than plaintext transactions. Fruitchains satisfy the following key property:

Theorem 6 (Fairness, [137]). *For any constant $0 < \delta < 1$, there exists a constant $T_0 = \Theta\left(\frac{1}{\delta}\right)$ such that the Fruitchains protocol is δ-approximately fair with respect to an adversary that controls at most β fraction of the computational power.*

Notice that fairness holds even without considering incentive mechanisms. To obtain incentive-compatibility, Fruitchains makes a key design decision, which is to distribute rewards (block rewards and transaction fees) evenly to the $T(\kappa)$ block miners preceding each block. Suppose that within any stretch of $T(\kappa)$ transactions, the total reward from transaction fees and block rewards is V. We say that honest mining is a *β^*-coalition-safe γ-Nash equilibrium* if, with overwhelming probability, no coalition of $\beta < \beta^*$ can gain more than a multiplicative factor $(1+\gamma)$ utility, regardless of the transactions being processed. Under this reward allocation scheme, Fruitchains satisfies the following property:

Remark 7.3.2 ([137]). *Consider a blockchain protocol that satisfies δ-approximate fairness with respect to β attackers. If rewards are allocated to the $T(\kappa)$ block miners preceding each block (including the block itself), honest behavior is a β-coalition-safe, 3δ-Nash equilibrium.*

This remark is straightforward to derive. Fairness implies that within each stretch of $T(\kappa)$ blocks, the fraction of adversarial blocks is upper bounded by $\beta(1+\delta)$, so the adversary's reward is upper

bounded by $\beta(1+\delta)V$. By fairness, if the adversary had followed the honest protocol, it would be guaranteed to reap a reward of at least $(1-\delta)\beta V$. Hence the adversary gains a factor of at most

$$\frac{(1+\delta)}{(1-\delta)} \leq 1+3\delta$$

in rewards for $\delta < \frac{1}{3}$. Notice that this result depends on two key components: first, the underlying blockchain protocol should be fair. Second, the reward allocation scheme should split rewards evenly among miners. By adopting a version of Fruitchains adapted to PoS (e.g., as proposed in Snow White [52] or Ouroboros [101]), we can obtain analogous results.

There is a key downside to sharing transaction fees across a sequence of block proposers, while also allowing users to propose their own fees. Since miners do not collect all the transaction fees from a block, they are less incentivized to include transactions with higher fees first; this means that users have a much weaker incentive to propose fees above the minimum. A key question that must be answered is whether such a system can raise enough transaction fee revenue to pay for network security. At the moment, there is no research quantifying this question (as far as we know).

7.3.2.3 Algorithmic fees, miner gets full transaction fees

A third operating point is that transaction fees could be chosen purely algorithmically, but miners would still reap the full transaction fees of each block they mine. To the best of our knowledge, there are no cryptocurrencies or papers advocating for this approach. Nonetheless, there are related workflows in existing systems. For instance, the Bitcoin Core wallet has a slider that allows the user to choose a desired processing speed; the wallet then algorithmically chooses a fee based on the blockchain's historical processing rates as a function of fees. Since the underlying blockchain is just Bitcoin, the miner that includes said transaction would reap the (algorithmically-chosen) fee.

The previous example belies a key point: our reason for exploring algorithmic fees in the first place is to prevent users from paying higher fees to enable faster transaction processing. This is the key source of congestion, as described in Section 7.3.1. A relevant question, then, is how an algorithm might choose a transaction fee.

Fundamentally, transaction fees pay for two properties: security and convenience. This cost can be broken down into the internal (computational and storage) cost of validating and storing transactions, and the external (communication, security) cost of coming to consensus with the rest of the network. Internal costs are primarily governed by transaction byte size; the larger the transaction, the more resources are required to store the transaction. External costs are more subtle to quantify. At first glance, these costs also seem to scale primarily with transaction byte size, since the consensus protocol does not explicitly differentiate between transactions and byte size determines communication cost. However, in practice, transaction *value* also plays an important role, for a few reasons.

1. **Processing cost.** High-value transactions require better security; they are more appealing targets, and therefore bear a higher standard for validation and finalization. Since it is common practice to wait longer before considering a high-value transaction as confirmed, the network implicitly expends more effort to confirm high-value transactions.[4]

2. **Security.** Taking a more global view, transaction fees in aggregate fund the security of the network as a whole. Since higher-value payments increase the need for security, they should also bear a larger burden in terms of paying for that increased security. User-proposed fees do

[4] In the language of Chapter 3, high-value transactions may require a smaller error probability ε.

not satisfy this requirement, because they are only tied to transaction value through the user's preferences.

3. **Economic fairness.** Finally, existing payment networks (e.g., credit cards) typically set fees as a fixed percentage of transaction value. If Unit-e's fees are lower than those of traditional payment systems, then it is capturing a surplus for users. In order to distribute that surplus fairly among users, transaction fees should similarly be set as a percentage of transaction value.

These arguments collectively suggest that transaction fees should somehow scale with transaction value (not necessarily linearly). We therefore consider a fee mechanism that is programmatically determined. For each transaction, wallet software will evaluate a function $w(\cdot)$ that takes as inputs a number of features, including transaction value v, transaction byte size b, and possibly other parameters capturing the current level of network health h. Including network health in the computation enables adaptive fees that respond to network conditions, such as congestion and/or gaps in security. Although we have not specified yet how w should be defined, it should satisfy a few basic properties:

- *Boundedness:* For all $v, b, h \geq 0$,

$$\underline{w} \leq w(v, b, h) \leq \bar{w}v,$$

  where $\underline{\text{w}}$ is a hard-coded lower bound on the transaction fee, chosen to make spamming attacks costly. Similarly, $\bar{w}$ is an upper bound on the fee *rate*, or the amount of fees per unit transaction value. We expect to choose $\bar{w}$ based on merchant feedback.

- *Monotonicity (transaction value):* For any $v_1 < v_2$, b, and h,

$$w(v_1, b, h) \leq w(v_2, b, h).$$

- *Monotonicity (transaction byte size):* For any $b_1 < b_2$, v, and h,

$$w(v, b_1, h) \leq w(v, b_2, h).$$

Many functions satisfy these broad requirements, with different tradeoffs. Consider the operating point where miners reap the full transaction fees of each block they mine; intuitively, for a given function w, the weaker its dependency on transaction value, the less congestion we expect to see. This is because of the arguments in Section 7.3.1; congestion is caused by miners choosing which transactions appear in each block. At the same time, weakening this dependency degrades the system's security properties. A key research question is to analyze the effects of different transaction fee functions on the congestion-security tradeoff.

In summary, the approach of choosing transaction fees purely algorithmically has some attractive properties; primarily, transaction fee rates would no longer be unbounded. However, the core problem of congestion remains. High-fee transactions will still be processed earlier, which promotes a hierarchy among transactions, and raises questions of fairness to users.

7.3.2.4 Algorithmic fees, decoupled rewards.

The fourth operating point consists of choosing fees purely algorithmically, while also partially decoupling the allocation of transaction fees from the block(s) that included those transactions. This is the least well-understood among the four operating points, but it has some appealing properties.

First, transaction fees can be bounded. Second, congestion can be reduced, by reducing the incentive for miners to strongly favor transactions with high fees. Third, unlike the operating point in Section 7.3.2.2 (user-selected fees, decoupled rewards), it does not matter that users are not incentivized to propose higher fees, since we can algorithmically choose fees that support network functionality. A key downside of this approach is complexity; analyzing the incentives of miners and the relation between the fee structure and network viability may be quite complicated if the transaction fee algorithm is complex. We view this as the main obstacle to this approach, and an important direction for future research and consideration in the transaction fee design of Unit-e.

7.4 Acknowledgement

The authors thank K. Ruan and G. Wang for collaborations on the design and analysis of the geometric incentive mechanism. This chapter includes material from [103] (Section 7.1) and [69] (Section 7.2).

Chapter 8

Privacy and Identity Management

Giulia Fanti, CMU
Andrew Miller, UIUC
Pramod Viswanath, UIUC

The notion of identity is critical to financial transactions. Customers must trust that they are sending money to the right entity, and merchants want to know the identities of their clients for practical reasons, such as shipping and regulatory compliance. However, with the rise of credit cards, a third party has entered the identity management picture: payment systems. Credit card providers inherently see the identities of participants in all transactions. While many users have grown accustomed to this, it requires a degree of trust in a third party (the payment system) that is not always warranted. If this trust is violated—e.g. in a data breach—the privacy fallout can be substantial. A key observation of cryptocurrencies is that payment systems *do not need to know the identities of their participants.* This has been true of many financial systems, such as cash and barter systems. In this vein, it is useful to ask how a cryptocurrency can facilitate strong identity management between relevant parties (i.e., merchants and customers), while cutting out unnecessary information leakage to middlemen.

We abstract the identity management problem in two parts: The first part involves identity management among physical participants, such as merchants and clients. We maintain that such interactions should be managed entirely at the *application layer.* For example, users can maintain internal, private databases of the account addresses (and physical entities) with whom they interact. Similarly, regulatory compliance checks (e.g., KYC/AML requirements) can be managed by applications sitting on top of the Unit-e stack. This abstraction is similar to cash; today, regulatory compliance for cash transactions is not enforced by the cash itself, but by the transacting parties. For instance, if one were to deposit a large amount of cash in a bank, it is the bank that executes regulatory compliance checks. Past transaction and identity data can be stored and maintained locally on users' devices, in apposite hardware, or on the cloud—these decisions are explicitly beyond the scope of a cryptocurrency's design. This identity layer can be operationally isolated from the underlying blockchain.

The second part deals with identity management in the blockchain: how do we ensure that users can transact with the correct parties without leaking that information to other blockchain participants? This is a technically much harder problem, and will be the focus of this section. The purpose of cryptocurrencies is transparency between participants; because of this, the blockchain data structure must be visible to all participants. This poses clear privacy risks, as the blockchain also contains every

transaction in the history of the currency. To mitigate the associated privacy risk, Bitcoin originally proposed using pseudonymous identifiers: an adversary observing the blockchain learns only the transaction patterns of each pseudonym. This paradigm has been adopted by most cryptocurrencies today. As long as these pseudonyms cannot be linked to the owner's human identity, privacy is preserved. However, multiple studies have shown that pseudonymity is vulnerable to deanonymization, particularly in the presence of side information [131, 22, 123, 21, 134]. Understanding how to build a truly privacy-preserving cryptocurrency is an open question.

In practice, there are two main ways for users to be deanonymized: through the blockchain, or through the network. Blockchain-based deanonymization attacks generally use a combination of patterns in blockchain transactions and side information (e.g., knowledge of the owners of certain wallets) to deanonymize users. The so-called *transaction graph* can be used to analyze transaction patterns between different public keys, and public keys held by the same person can often be linked [123, 21, 134]. Worst case, this can enable an attacker to associate a pseudonym with a human identity. The second class of deanonymization attacks is run by adversaries who observe traffic flowing over the P2P network. In most cryptocurrencies, new transactions are spread over the network according to some pre-specified flooding protocol (typically a gossip algorithm). If an adversary observes this traffic at a fine enough time resolution—e.g., by setting up a supernode that connects to many nodes—it can often link transactions (and hence the address of the sender) to the IP address that originated the transaction. This is a less severe deanonymization attack than blockchain-based attacks, but it is nonetheless significant, and often-ignored [71]. We are interested in both classes of attacks.

This chapter discusses two main components of Unit-e's privacy roadmap:

1. We briefly summarize a powerful technique for providing transaction privacy at the blockchain layer known as *zk-SNARKs*. The key idea behind zk-SNARKs is to encrypt transactions in such a way that users can verify their validity without learning anything about the contents of the transaction. zk-SNARKs are the technical foundation of the privacy-preserving cryptocurrency ZCash; however, they have historically faced scalability challenges. We describe recent advances in the scalability of zk-SNARKs that may enable their use in Unit-e. Although the Unit-e research corpus has not innovated on this front yet, we view blockchain-level privacy as a major direction for upcoming research.

2. We summarize our recent work on providing network-level privacy through a system called Dandelion. Intuitively, Dandelion gossips transactions in a randomized fashion that is designed to prevent adversaries from launching network attacks. Dandelion exhibits strong theoretical privacy guarantees while also incurring low latency overhead. Dandelion will also be integrated into Unit-e's networking stack.

We begin with a discussion of blockchain-level privacy, and a primer on zk-SNARKs in Section 8.1. Then, we move on to network-level privacy and countermeasures in Section 8.2.

8.1 Blockchain-Level Privacy

We have seen that blockchains do not necessarily provide strong privacy, and the transaction graph in Bitcoin, for example, is a source of information that can trace a user's activities. Cryptography, and in particular, zero knowledge proofs, can come to the rescue, enabling users to keep their confidential information encrypted on the blockchain ("hidden in plain sight"), while still ensuring system-wise guarantees such as conservation of money. In this section, we provide an introduction to zero-knowledge proofs, and summarize some promising technologies that are being evaluated for

Unit-e's privacy roadmap. Since we view blockchain-level privacy as critical for a payment system, this topic will be a major focus of our future work.

In short, a zero knowledge proof allows a prover to convince a verifier of some claim, without revealing confidential information associated with that claim. An example of such a claim is

> "I know the preimage of hash h."

Clearly you can prove this by revealing x, since anyone can then verify $\mathcal{H}(x) \stackrel{?}{=} h$. With a zero knowledge proof scheme, you can prove the claim while keeping x secret. To see how this ability is useful for constructing a cryptocurrency, consider the following claim:

> "I am authorized to spend a coin, and this coin has not yet spent."

Given such a proof scheme, a user could spend a coin without revealing where the coin came from, effectively making them all fungible. The idea of using such a zero knowledge proof scheme as the basis for a digital currency dates back to 1999 [155], and forms the foundation of cryptocurrencies like Zcash and Monero.

There are a wide range of zero knowledge proof schemes, applicable to different kinds of claims, and with a variety of performance tradeoffs. This remains an active area of research. In the rest of this section, we outline several leading approaches to zero knowledge proofs and their application to cryptocurrencies.

8.1.1 Zero Knowledge Schemes

A zero knowledge proof scheme can be summarized in Camenisch-Stadler notation. This specifies the public information (the statement), which is known to the prover and verifier, the private information (the witness), which is known only to the prover, and the nature of the claimed relationship between them (the predicate P):

$$ZK\{(w) : P(x, w) = 1\}$$

In language theoretic terms, the predicate P defines a language $\mathcal{L}_P$, where a statement x is in the language if and only if there exists a w such that $P(x, w) = 1$.

A zero knowledge protocol provides the following three subroutines:

$$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, P)$$

$$\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w)$$

$$\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{pp}, x, \pi)$$

where κ denotes a security parameter. The Setup routine takes a description of the predicate, P, typically expressed as an arithmetic circuit, or other representation depending on the protocol. The resulting public parameters pp are used for proving and verifying. The Prove routine, when given a valid witness w, produces a π that Verify accepts.

In order to make asymptotic security claims, we must introduce a security parameter $\kappa \in \mathbb{N}$. The running time of the protocol will grow slowly (at most polynomially) with κ, while the success probability for a computationally bounded attacker is negligible ($\mathsf{negl}(\kappa)$). In case the predicate P depends on the security parameter, we say P is an element of a family of predicates $\{P_\kappa\}_{\kappa \in \mathbb{N}}$, where the time to compute P is polynomial in κ.

Security properties. A zero knowledge proof scheme provides two main desired security guarantees, which we state informally, eliding many details and caveats:

- *(Completeness.)* An honest prover with a valid witness can always convince an honest verifier. More formally, for any (x, w) such that $P(x, w) = 1$, we have

$$\mathbb{P}\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa),\\ \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w) :\\ \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1\end{array}\right] = 1 - \mathsf{negl}(\kappa)$$

  where the probability is taken over randomness in the first two subroutines.

- *(Proof-of-Knowledge.)* The only way to produce a valid proof π that Verify accepts, is if the prover actually "knows" a valid witness w. Knowledge of this kind is difficult to define, typically through *extractability*.

  We say that a scheme is extractible if, for any x, and for any probabilistic polynomial time adversary $\mathcal{A}$, there exists an extractor $\chi_{\mathcal{A}}$ such that:

$$\mathbb{P}\left[\begin{array}{ll}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa, P), & P(x, w) \neq 1\\ ((x, \pi); w) \leftarrow (\mathcal{A} \| \chi_{\mathcal{A}})(\mathsf{pp}): & \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1\end{array}\right] = \mathsf{negl}(\kappa)$$

  Extractability is a strong property, and in particular, it implies *soundness*, meaning that the prover can only convince the verifier of true statements, i.e. $x \in \mathcal{L}_P$.

- *(Zero-knowledge.)* The proof π does not reveal anything about the witness w. There can be several subtle variations, typically defined through simulation: anything the verifier can compute after seeing π, it could also have simulated without π.

  More formally, there exists a simulator setup routine $(\mathsf{pp}, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa, P)$ such that pp is distributed identically as to the original Setup, but additionally outputs a trapdoor τ. The trapdoor is used by a simulator routine SimProve, which does *not* know the witness but nonetheless must produce a valid proof.

$$\left|\begin{array}{l}\mathbb{P}\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa),\\ \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w) :\\ \mathcal{A}(\mathsf{pp}, x, \pi) = 1\end{array}\right] -\\ \mathbb{P}\left[\begin{array}{l}(\mathsf{pp}, \tau) \leftarrow \mathsf{SimSetup}(1^\kappa, x),\\ \widetilde{\pi} \leftarrow \mathsf{SimProve}(\mathsf{pp}, x, \tau) :\\ \mathcal{A}(\mathsf{pp}, x, \widetilde{\pi}) = 1\end{array}\right]\end{array}\right| = \mathsf{negl}(\kappa)$$

We have focused our discussion on non-interactive proofs, as these are most naturally applied in a cryptocurrency setting. However, we point out that there are other variations involving interaction between the prover and verifier. One desired property of interactive proofs is "designated verifier," in which the proof is effectively non-transferable.

We also observe that not all schemes require public parameters. In some cases the public parameters are uniform random strings, in which case the Setup can be conducted in a transparent way, using random beacon values. However, if the Setup routine relies on drawing private random coins which are not included in the parameters, this requires a trusted setup.

Performance characteristics. We are especially interested in a handful of performance properties. These primarily depend on the size of the predicate $|P|$ in the given representation, e.g. the number of multiplication gates in an arithmetic circuit.

- (Prover time.) Preparing a proof is typically at least as long as evaluating the predicate, $O(\mathsf{poly}|P|)$.
- (Proof size.) A protocol is "succinct" if the proof size is constant with respect to $|P|$.
- (Verification time.) In a blockchain application, verification time is often the bottleneck, since a proof may be generated once but verified many times by independent validators.

8.1.2 Classes of Proof Schemes

We next describe several proof schemes that are commonly used in cryptocurrencies, and highlight their performance tradeoffs. We omit many variations as well, such as those based on database queries like vSQL [190], ZQL [77].

Discrete Log Proofs. The most well known zero knowledge proof schemes, which we refer to broadly as discrete log proofs [73, 158, 72], take place in a cryptographic setting with a group $\mathcal{G}$ in which the discrete log problem is hard. For example, the following denotes a proof that the prover knows how to open a given Pedersen commitment C, where g and h are random generators of $\mathcal{G}$:

$$ZK\{(x, r) : C = g^x h^r\}$$

There are generic ways to compose such proofs, for example using AND and OR combinators of expressions [38, 122]. In general, the proof size and the verification time depend on the predicate size $O(|P|)$.

An advantage is that the setup only needs to select random elements as generators, e.g., g and h, where the discrete log between them is not known. These can be obtained from public random beacon values, or by using a hash function, such as a "Nothing Up My Sleeve" number.

Recently, BulletProofs have been developed. The proof size is small, $O(\log |P|)$ for a circuit of size $|P|$, although verification takes longer.

CRS-based SNARKs. Many practical schemes, including those known as SNARKs, are able to achieve very small proof size and verification cost, by relying on a Common Reference String (CRS) which is sampled at setup time, through a procedure known as a "trusted setup."

For example, one component of the trusted setup used in Zcash involves generating a sequence of powers, $\mathsf{pp} = (..., g, g^\tau, g^{\tau^2}, ..., g^{\tau^k})$ up to some degree k (called the "powers of tau"). The value τ is randomly sampled, but is omitted from the final public parameters, assumed to be destroyed after running Setup. As part of the Prove routine, the prover uses these public parameters to evaluate a degree-k polynomial ϕ on τ "in the exponent," i.e. to compute $g^{\phi(\tau)}$. However, the prover is not able to compute $g^{\tau^{k+1}}$, and hence cannot evaluate polynomials of any larger degree. If the trusted setup fails, such that the prover learns τ, then he may be able to forge proofs of false statements.

MPC-in-the-Head. The main idea of MPC-in-the-Head is to make use of existing protocols for multiparty computation (MPC), but repurpose them as a zero knowledge proof. In ordinary MPC [87, 84], a protocol is executed jointly among several nodes, and privacy is guaranteed as long as one node keeps its data secret. To adapt an MPC protocol to the zero knowledge proof

Table 8.1: Summary of Performance Characteristics of Generic ZK Proofs

	Proving Cost	Proof Size	Verification Time	Trusted Setup
Discrete Log	$\tilde{O}(\|P\|)$	$O(\|P\|)$	$O(\|P\|)$	No
BulletProofs	$\tilde{O}(\|P\|)$	$O(\log \|P\|)$	$O(\|P\|)$	No
MPC-in-the-Head	$\tilde{O}(\|P\|)$	$O(\sqrt{\|P\|})$	$O(\sqrt{\|P\|})$	No
STARKs	$\tilde{O}(\|P\|)$	$O(\log \|P\|)$	$O(\log \|P\|)$	No
zkSNARKs	$\tilde{O}(\|P\|)$	$O(1)$	$O(1)$	Yes

setting, a prover simulates the execution of an entire MPC protocol, playing all the roles himself, and committing to a transcript of every role's state and actions. At the end of the protocol, the prover reveals $N-1$ of the transcripts, providing evidence that the protocol was carried out correctly, while the final N^{th} transcript is destroyed to ensure privacy.

Examples include ZKBoo [81], ZKB++ [43], and Ligero [19]. So far, this approach is able to achieve sublinear proof size and verification time, e.g. a proof size and verification cost of $O(\sqrt{(|P|)})$ for a circuit of size $|P|$.

PCP-based STARKs. Recent work [28] has introduced a class of protocols called STARKs, which are based on probabilistically checkable proofs.

Summary and open questions. We summarize the performance achieved by the generic zero knowledge proof schemes in Table 8.1. zk-SNARKs are, so far, the best performing among the generic proof schemes, although their main disadvantage is the reliance on a trusted setup. For this reason, Zcash makes use of a multiparty ceremony to generate the trusted setup in a distributed way [35, 29]. STARKs, MPC-in-the-head, and Bullet Proofs all accept performance tradeoffs in exchange for avoiding the trusted setup. Since efficiency is critical in Unit-e, we are currently evaluating the use of zk-SNARKs for transaction privacy. This may change as new developments emerge in this space; a key open question is whether it is possible to obtain *succinct* proofs, with constant verification time, without any trusted setup.

8.2 Network-Level Privacy

Network-level transaction deanonymization has become a significant concern in recent years [31, 107]. The core idea is that an adversary can strategically place itself in the network topology to infer properties of the source IP address for a given transaction. This is a privacy concern because IP addresses can, in some cases, be used to link a transaction to the person who originated that transaction. Moreover, these kinds of attacks are relatively cheap and easy to run.

This problem is well-known in the cryptocurrency community, and several potential solutions have been proposed. Our goal in this section is to discuss existing solutions, provide lower bounds on what is possible, and highlight the tradeoffs. Key questions include understanding the severity of attacks against existing broadcasting mechanisms and designing networking protocols that provably prevent user deanonymization. We also present our proposed solution to this problem, **Dandelion**, which is a lightweight network anonymity algorithm with theoretical privacy guarantees.

First-order solutions. Many users of Bitcoin and other cryptocurrencies adopt third-party routing tools (e.g., Tor or VPNs) to mitigate the threat of network deanonymization [32]. Such solutions

only benefit the individuals that adopt them; users who are unaware of network privacy threats—or not technically sophisticated enough to use these tools—remain unprotected. We maintain that a digital payment system should implement *network-wide* privacy protection mechanisms.

A natural network-wide solution might be to integrate Tor or VPN support into client software. At least nominally, this approach addresses the problem at hand. One potential challenge is that system designers may be reluctant to tie the network's privacy guarantees to a third-party system. Moreover, if the cryptocurrency eventually supports a substantial amount of traffic, third-party services may be unwilling to route that traffic without compensation.

An extension of this idea is to incorporate privacy-preserving routing (e.g. onion-routing) into the cryptocurrency's own P2P networking stack. This approach also solves the privacy problem, and Monero is doing precisely this with I2P routing [6]. One challenge is that such implementations require substantial developer effort and expertise due to the difficulty of implementing cryptographic protocols in a secure fashion. This section will focus primarily on adding privacy-preserving routing to the P2P network; however, we will focus on privacy-preserving routing schemes that are more lightweight than Tor or I2P in that they do not involve cryptography. Notice that the previous section focused entirely on cryptographic primitives; to the best of our knowledge, meaningful blockchain-level privacy cannot be achieved without cryptography. However, at the network level, there exist intermediate solutions that give weaker guarantees than cryptographic protocols, at a fraction of the development cost.

We will focus in particular on commonly-studied models and a summary of results and open questions. We begin by presenting relevant prior literature on anonymous communication, and highlighting the gaps between this literature and the needs of cryptocurrencies. Then, we will present a candidate solution, Dandelion.

8.2.1 Models

Work in this space typically models the underlying network as a graph $G(V, E)$. In the context of cryptocurrencies, V denotes the set of all nodes and E is the set of edges representing TCP connections between nodes. Many papers consider the problem of deanonymizing a single transaction from a single node [160, 71, 31, 107]. In this context, we let $v^* \in V$ denote the true source node. The adversary's goal is to guess v^* given some observations of a transaction's spread, where the transaction is drawn from message space $\mathcal{X}$. Other papers have considered the problem of *joint deanonymization*, in which multiple transactions are mapped to multiple users [34, 70]. To model this scenario, researchers have assumed that each node $v \in V$ generates one transaction [34]. In this case, we use $X_v \in \mathcal{X}$ to denote the set of messages generated by node $v \in V$. Notice that one node can generate multiple transactions, but we will assume in the following that X_v is a single transaction or message.

8.2.1.1 Adversarial Model

In network deanonymization literature, the adversarial model is one of the main variants that changes between papers. This model affects what information the source estimator can use, which substantially impacts the accuracy of the estimator. In the context of deanonymization, researchers and practitioners have considered two main adversarial models: *eavesdropper adversaries* and *botnet (spy-based) adversaries*. Eavesdropper adversaries run a supernode that connects to all (or a substantial subset) of nodes in the network (Figure 8.1). Critically, from the perspective of an honest node, the eavesdropper looks just like any other node. Honest nodes therefore relay transactions normally, allowing the eavesdropper to collect timestamps and other metadata. Combined with

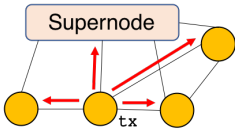


Figure 8.1: Eavesdropper adversary. A well-connected supernode eavesdrops on relayed communications.

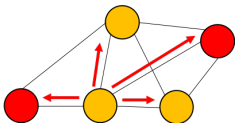


Figure 8.2: Botnet adversary. Red nodes represent corrupt "spy nodes", which use observed metadata to infer the transaction source.

information about the graph topology, this metadata can be used to infer the source of a particular transaction. One important property of eavesdropper nodes is that they typically do not relay messages; they only collect communications relayed by other nodes.

Botnet or spy-based adversaries (Figure 8.2) instead consist of a set of corrupt, colluding nodes that participate in the network normally, both accepting *and* relaying information. We let p denote the fraction of spies in the network, and let $V_A, V_H \subseteq V$ denote the set of adversarial and honest nodes, respectively. As the name suggests, this adversarial model is motivated by a botnet that spawns cryptocurrency nodes. The key difference compared to the eavesdropper adversary is that each botnet node may have limited visibility into the network (e.g., if it only connects to a few peers rather than the entire network), and botnet adversaries may inject packets into the network.

These models are caricatures, and in practice, an adversary may behave like a combination of the two models, or something else entirely. However, these models are informed by a long literature on anonymous communication, as well as practical attacks on cryptocurrencies. As such, defending against them is an important first step in hardening the networking stack of cryptocurrencies against deanonymization attacks.

8.2.1.2 Metrics

Let $A \in \mathcal{A}$ denote the information visible to the adversary at the time of inference, where $\mathcal{A}$ denotes the space over which such data is defined. Notice that the definition of $\mathcal{A}$ changes between adversarial models; for example, in the spy-based model, it might represent the timestamps at which each spy received each transaction message. In the case where the adversary is trying to deanonymize a single transaction, the adversary's goal is to find an estimator function $E^* : A \to V$, which outputs a source estimate $\hat{v} := E^*(A)$ such that

$$E^* = \underset{E:A\to V}{\arg\max} \quad \mathcal{P}(\hat{v} = v^*), \tag{8.1}$$

where the probability is taken over the spreading pattern, the source node v^*, the adversary's observations A, and any randomness in the estimator itself. $\mathcal{P}(\hat{v} = v^*)$ is the adversary's *probability of detection*, which is a key metric of interest.

In models where each user generates one or more transactions and the adversary conducts a joint deanonymization, the correct anonymity metric is less obvious. One option, used in [34, 70], is to consider the adversary's mapping $M : (\mathcal{X}, \mathcal{A}) \to V$ from each transaction to a node. For simplicity of notation, we remove the dependency on A in the following. The user's anonymity is evaluated by measuring the quality of this mapping, using the notions of *precision* and *recall*.

Precision and recall are performance metrics commonly used in information retrieval for binary classification. Suppose we have n data items, each associated with a class: 0 or 1. We are given a classifier that labels each data item as either a 0 or a 1, without access to the ground truth. We designate one of these classes (e.g. class 1) 'positive'. For a given classifier output on a single item,

a *true positive* means the item was correctly assigned to class 1, and a *true negative* means the item was correctly assigned to class 0. A *false positive* means a 0 item was incorrectly classified as a 1, and a *false negative* means a 1 item was incorrectly classified as a 0. If we run this classifier on all n data items, precision and recall are defined as follows:

$$\begin{aligned} \textbf{Precision} &= \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Positives}|} \\ \textbf{Recall} &= \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Negatives}|} \end{aligned}$$

where $|\cdot|$ denotes the cardinality of a set, and 'True Positives' denotes the set of all data items whose classification output was a true positive (and so forth).

Precision can be interpreted as the probability that a randomly-selected item with label 1 is correct, whereas recall can be interpreted as the probability that a randomly-selected data item from class 1 is correctly classified. Adapting this terminology to our problem, we have a multiclass classification problem; each server is a class, and each transaction is to be classified. For a given node $v \in V$ and mapping M, the precision $D_M(v)$ comparing class v to all other classes is computed as[1]

$$D_M(v) = \frac{\mathbb{1}\{M(X_v) = v\}}{\sum_{w \in V} \mathbb{1}\{M(X_w) = v\}}, \tag{8.2}$$

and recall is computed as

$$R_M(v) = \mathbb{1}\{M(X_v) = v\} \tag{8.3}$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. In multiclass classification settings, precision and recall are often aggregated through *macro-averaging*, which consists of averaging precision/recall across classes. This approach is typically used when the number of items in each class is equal [159]. One can therefore average the precision and recall over all servers and take an expectation, giving an expected macro-averaged precision of $\mathbf{D} = \mathbb{E}[D_M] = \frac{1}{\tilde{n}} \sum_{v \in V_H} \mathbb{E}[D_M(v)]$ and recall of $\mathbf{R} = \mathbb{E}[R_M] = \frac{1}{\tilde{n}} \sum_{v \in V_H} \mathbb{E}[R_M(v)]$. Notice that in expectation, recall is equivalent to probability of detection.

8.2.2 Related Work

Prior literature has examined two main relevant questions: how vulnerable are existing protocols to deanonymization attacks, and can we design protections for them? Both questions have received substantial attention, both in the context of cryptocurrencies and in a more abstract sense. We begin with a summary of research on detecting the source of randomized spreading processes. Then we overview the literature on anonymous communication, both in point-to-point and one-to-many communication settings. The take-home message of this review is twofold. First, the broadcasting primitives used in current cryptocurrencies are vulnerable to deanonymization. Second, the classical literature on privacy-preserving communication in P2P networks does not sufficiently address the problem. In particular, prior work has focused only minimizing probability of detection; this is a weaker metric than jointly considering both precision and recall.

8.2.2.1 Source Detection

Many cryptocurrencies—including Bitcoin—broadcast content using a gossip protocol called *diffusion*. Under diffusion, each node transmits the message to each of its neighbors with independent,

[1]Following convention we define $D_M(v) = 0$ if both the numerator and denominator are 0 in Equation (8.2).

exponentially-distributed delays. The problem of detecting the source of a diffusion process over a graph is well-studied. Shah and Zaman studied this problem on infinite regular tree topologies, under a so-called *snapshot model* [160]. Under this model, the diffusion process is allowed to spread until some time T. At this time, the adversary gets to observe which nodes have the message, and which do not. Surprisingly, the authors showed that one can reliably infer the source of a diffusion process, even as time grows unboundedly [160]. That is, there exists a positive constant μ such that the probability of detection satisfies:

$$\lim_{T\to\infty} \max_{E} \quad \mathcal{P}(\hat{v} = v^*) \geq \mu > 0.$$

Shah and Zaman later extended these results to random, irregular trees [161], and other authors studied heuristic source detection methods on general graphs [74, 144, 117] and related theoretical limits [182, 125, 99].

Although the snapshot adversarial model analyzed in this work does not generally arise in cryptocurrencies, this research led to a number of papers that consider the spy-based adversary from section 8.2.1. For example, Pinto et al. considered a spy-based adversary that observes a diffusion process where the delays are truncated Gaussian random variables [142], and Zhu and Ying consider a spy-based adversary with standard exponential delays [192]. These papers do not theoretically characterize the maximum-likelihood probability of detection, but they do propose efficient heuristics that perform well in practice. Fanti and Viswanath also study diffusion under an eavesdropper adversary [71], which has been seen in practical attacks on the Bitcoin network [31, 107]. [71] shows that an eavesdropper observing a diffusion process on a regular tree can detect the source with probability bounded away from zero as the degree of the underlying tree tends to infinity. Practically, these deanonymization efforts revolve around the notion of centrality; because diffusion spreads content symmetrically on the underlying network, the message (roughly) spreads in a disc over the graph, with the true source at the middle of that disc.

The above results imply that an adversary with partial global oversight can infer the shape of the disc and identify the central node with non-negligible probability. These results suggest that diffusion is poorly-suited to protecting users' anonymity at the network level, which motivates the need for alternative spreading protocols that protect users' anonymity.

8.2.2.2 Anonymous Communication

Anonymous communication for P2P networks has been an active research area for decades. Most systems proposed in this space rely on two key ideas: randomized routing (e.g., onion routing, Chaumian mixes) and dining cryptographer (DC) networks. Although both approaches can be used for the multicasting/broadcasting primitives required in cryptocurrencies, DC networks are specifically designed for anonymous broadcasting, whereas randomized routing protocols are designed for point-to-point communication. As a result, systems that use DC nets [44] are typically designed for broadcast communication applications, such as microblogging [48]. At a high level, the idea behind DC nets is that all nodes should broadcast messages encrypted with correlated one-time pads. The sum (across all nodes) of these one-time pads is designed to cancel out; assuming only one node broadcasts a real message (instead of a 0), the sum of the ciphertexts will reveal that message; at the same time, the identity of the author cannot be recovered. DC nets are known to be fragile and inefficient, requiring $\Theta(n)$ communication per communicated bit [88]. For example, if two nodes broadcast a message at the same time, their messages get summed in the output in such a way that neither message can be recovered. Because of this property, if even one node misbehaves, the entire result can be invalidated. Several authors have proposed modifications of DC nets that address these challenges [83, 88, 188, 48, 185]. A key idea from this literature is to use many small DC nets,

rather than a single one for the entire network. This idea mirrors the reasoning behind blockchain sharding; consensus is fundamentally not scalable, so we instead solve multiple smaller sub-problems. Despite notable innovations in this area, DC networks have not yet become scalable enough to enjoy widespread adoption in practice.

Systems based on randomized routing are generally more efficient than those based on DC nets, though randomized routing is tailored for point-to-point communication. The core idea of most randomized routing is that messages traverse a random walk over an underlying network; this random walk is implemented such that intermediate nodes in the random walk cannot learn the source and/or destination. A natural approach for adapting randomized routing to the broadcast setting is to execute it for a (possibly random) number of hops before initiating a regular, non-anonymous broadcasting process. This approach matches the *de facto* spreading pattern of Bitcoin users who connect their clients to the network via Tor, for example. In the area of point-to-point randomized routing, early works like Crowds [148], Tarzan [78], and P5 [163] paved the way for later practical systems, such as Tor [60] and I2P [189], which are used both within cryptocurrencies and in the Internet ecosystem at large. More recently, related proposals have targeted specific application spaces and/or recent anonymity tools, including Drac [54], Pisces [126], and Vuvuzela [176]. A different randomized routing approach is called adaptive diffusion (AD) [66], which is specifically designed for one-to-all message broadcasting. AD is a synchronous protocol that spreads on a tree topology that is overlaid on the existing P2P network. Despite strong theoretical properties on restrictive graph topologies (e.g., infinite regular trees), AD requires substantial coordination and can 'get stuck' on real graphs, meaning that some messages do not reach the entire network [66]. This property is unacceptable in cryptocurrencies: all nodes should receive all messages for fairness and consistency purposes.

8.2.3 Lower Bounds

Given the substantial work on this topic, it is natural to ask what privacy guarantees are fundamentally possible without resorting to cryptographic protocols. Many of the systems discussed in Section 8.2.2 include theoretical analysis, but very few provide optimality guarantees. Moreover, the ones that do come with optimality guarantees apply only to per-user metrics, such as probability of linkage [148]. This metric overlooks the fact that adversaries can use data from many users to execute *joint deanonymization*.

To understand the effects of joint deanonymization, we need lower bounds on precision and recall. In recent work [34], we show the following lower bound on the expected recall (probability of detection) $\mathbf{R}_{\texttt{OPT}}$ and precision $\mathbf{D}_{\texttt{OPT}}$ of any randomized routing scheme using a recall- or precision-optimal estimator.

Theorem 7 ([34])**.** *The optimal precision and recall on a network with a fraction p of adversaries and any spreading policy are lower bounded as*

$$\mathbf{D}_{\texttt{OPT}} \geq p^2 \tag{8.4}$$

$$\mathbf{R}_{\texttt{OPT}} \geq p. \tag{8.5}$$

The proof can be found in Section 8.2.5.1. This lower bound gives a point of comparison for proposed protocols. For example, although we do not include the analysis here, it is possible to show that the Crowds protocol [148] achieves suboptimal precision, even though Danezis *et al.* show that it achieves optimal recall [53]. Indeed, to the best of our knowledge, none of the protocols described in the related work achieve optimality in both precision and recall. In the following section, we discuss a protocol that achieves within a logarithmic factor of these lower bounds.

8.2.4 Dandelion

We recently proposed Dandelion, an anonymous transaction propagation protocol for cryptocurrency networks [34, 70] that achieves (near-)optimal precision and recall guarantees. Moreover, much of the related work in Section 8.2.2 was mostly designed for general peer-to-peer applications; it does not account for cryptocurrency-specific details, such as typical P2P topologies, networking DoS protections, etc. Dandelion was designed with Bitcoin in mind, though the underlying ideas apply to many other cryptocurrencies as well. Indeed, Unit-e will incorporate Dandelion in its own networking stack.

We begin with a brief overview of Dandelion and its guarantees. The basic idea of Dandelion is to propagate each transaction in a random walk over the underlying graph; this random walk is called the *stem phase*. After a random number of hops, the stem phase ends and the transaction is broadcast via diffusion to the rest of the graph; this is called the *fluff phase*. This simple description hides a number of details that affect the scheme's anonymity properties. Dandelion proceeds in asynchronous epochs; each node advances its epoch when its internal clock reaches a random threshold (in practice, this will be on the order of minutes). Within an epoch, the main algorithmic components of Dandelion are:

(1) Anonymity graph: The random walk takes place on an overlay of the P2P graph called the *anonymity graph*. This overlay should be chosen either as a random cycle graph (i.e., a 2-regular graph) or a 4-regular graph. This 4-regular graph is embedded in the underlying P2P graph by having each node choose (up to) two of its outbound edges, without replacement, uniformly at random as Dandelion relays. This does not produce an exact 4-regular graph, but an approximation. Each time a node transitions to the next epoch, it selects fresh Dandelion relays.

(2) Forwarding of a node's own transactions: Each time a node generates a transaction, it forwards the transaction in stem phase along the *same* randomly-selected outbound edge on the anonymity graph. If the anonymity graph is a line graph, choosing this outbound edge is trivial; otherwise, the node must choose one of its outbound edges [70].

(3) Relaying of other nodes' transactions: Each time a node receives a stem-phase transaction from another node, it either relays the transaction or diffuses it. The choice to diffuse transactions is pseudorandom, and is computed from a hash of the node's own identity and epoch number. Note that the decision to diffuse does not depend on the transaction itself—in each epoch, a node is either a diffuser or a relay node for *all* relayed transactions. If the node is not a diffuser in this epoch (i.e., it is a relayer), then it relays transactions pseudorandomly; each node maps each of its incoming edges in the anonymity graph to an outbound edge in the anonymity graph (with replacement). This mapping is selected at the beginning of each epoch, and determines how transactions are relayed.

(4) Robustness mechanism: Each node tracks, for each stem-phase transaction that it sends or relays, whether the transaction is seen again as a fluff-phase transaction within some random amount of time. If not, the node starts to diffuse the transaction.

For more details on the implementation of Dandelion, we refer the reader to our follow-up paper, which includes a more sophisticated adversarial model as well as practical implementation considerations [70]. Under a spy-based, honest-but-curious adversarial model with a fraction p of spy nodes, Dandelion has optimal (up to a logarithmic factor) precision and recall.

Theorem 8 ([34])**.** *The expected recall ($\mathbf{R}_{\texttt{OPT}}$) and precision ($\mathbf{D}_{\texttt{OPT}}$) of Dandelion (i.e., Dandelion spreading on a line graph) with n nodes and a fraction $p < 1/3$ of adversaries, is upper bounded*

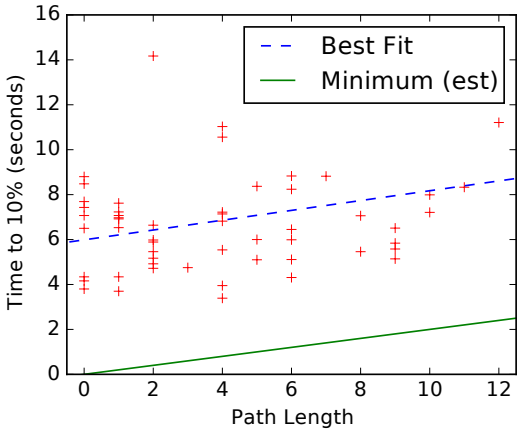


Figure 8.3: (Figure replicated from [70]) Time for a Dandelion transaction to propagate to 10% of the network as a function of the path length. Blue is the line of best fit at 218ms per hop.

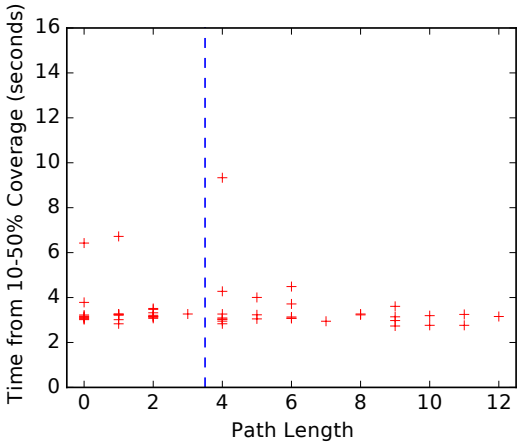


Figure 8.4: (Figure replicated from [70]) Time for a Dandelion transaction to propagate from 10% to 50% of the network.

by

$$\mathbf{R}_{\texttt{OPT}} \leq p + O\left(\frac{1}{n}\right) \tag{8.6}$$

$$\mathbf{D}_{\texttt{OPT}} \leq \frac{2p^2}{1-p}\log\left(\frac{2}{p}\right) + O\left(\frac{1}{n}\right). \tag{8.7}$$

The proof can be found in Section 8.2.5.2. Notice that the results in Theorem 8 achieve the lower bounds in Theorem 7 for recall and are within a logarithmic factor of optimal for precision. Of course, a more meaningful question is to bound precision and recall under a Byzantine adversary; this question is addressed in [70], where it is shown that Byzantine adversaries do not increase recall, and they increase precision by a factor of $O(\frac{1}{p\log(1/p)})$ [70].

Practical Tradeoffs. Dandelion's privacy guarantees come at the expense of transaction latency. A longer stem phase improves privacy but also increases the time required for a transaction to reach the whole network. To evaluate this tradeoff, in [70], we implemented Dandelion on a fork of Bitcoin Core, and deployed 30 nodes on the Bitcoin main network (mainnet). We then measured the latency of Dandelion transactions. Figure 8.3 shows the time required for a Dandelion transaction to reach 10% of the network, as a function of the length of the stem phase (in hops). The main takeaway is that longer stems increase latency by a factor of about 2 seconds per 10 hops, on average. Although [70] suggests using an average stem length of 10 hops in Bitcoin (where block times are on the order of 10 minutes), a low-latency payment system like Unit-e will likely need to adopt shorter stems, on the order of 3-4 hops. Figure 8.4 shows the time for a Dandelion transaction to spread from 10% of the network to 50%. This figure suggests that after the initial 10%, Dandelion does not affect transaction latency in a statistically significant way, which is expected behavior.

8.2.5 Proofs

The current section contains proofs of the main claims in the chapter; it can be safely skipped by readers who are not interested in the mathematical detail of the claims.

8.2.5.1 Proof of Theorem 7

For each honest server node v, we let S_v denote the set of (transaction, receiving spy node, timestamp) tuples $(x, u, T_u(x))$ such that transaction x was forwarded by honest node v to adversary $u \in V_A$ at time $T_u(x)$; $\mathbf{S}$ is the vector of all S_v's. Recall that for honest server v, the tuple $(x, u, T_u(x))$ is contained in S_v if v forwards message x to adversarial node u at time $T_u(x)$. Let us now define a related quantity $\bar{S}_v$ to denote the set of messages $x \in \mathcal{X}$ forwarded by v to some adversary such that x was not received by any adversarial node previously. This quantity $\bar{S}_v$ is useful in analyses involving the first-spy estimator. $\bar{\mathbf{S}}$ denotes the vector of all $\bar{S}_v$'s.

The following lemma implies a lower bound on recall by analyzing the *first-spy* estimator $E_{\texttt{FS}}$, which is defined for a particular transaction x as follows:

$$E_{\texttt{FS}}(A, x) := \arg\min_{v \in V}\{T_v(x)\}.$$

We use $\mathbf{R}_{\texttt{FS}}(v)$ and $\mathbf{D}_{\texttt{FS}}(v)$ to denote the first-spy estimator's expected recall and precision, respectively, for messages originating from node v.

Lemma 8.2.1. *If $v \in V_H$ is a honest server node in a network with a fraction p of adversaries, then the recall of the first-spy estimator is $\mathbf{R}_{\texttt{FS}}(v) = \mathbb{P}(X_v \in \bar{S}_v) \geq p$.*

Proof. Let U denote the node to which v first sends its message X_v, and let V_A denote the set of adversarial spy nodes. Then,

$$\begin{aligned}\mathbb{P}(U \in V_A) &= \sum_{u \in V, u \neq v} \mathbb{P}(U = u)\mathbb{P}(U \in V_A | U = u) \\ &= \sum_{u \in V, u \neq v} \frac{1}{n-1}\mathbb{P}(U \in V_A | U = u) = \frac{np}{n-1} \geq p, \end{aligned} \tag{8.8}$$

due to uniform distribution among the remaining nodes $V \backslash \{v\}$. Therefore we have,

$$\mathbb{P}(X_v \in \bar{S}_v) \geq \mathbb{P}(U \in V_A) \geq p. \tag{8.9}$$

Thus v's message is contained in $\bar{S}_v$ with probability at least p. The case where v simultaneously broadcasts X_v to multiple nodes can also be similarly bounded as above, and hence the lemma follows. □

To show (8.5), note that $\mathbf{R}_{\texttt{OPT}} \geq \mathbf{R}_{\texttt{FS}}(v) \geq p$, by Lemma 8.2.1. Next, we show that the first-spy estimator also has a precision of at least p^2 regardless of the topology or spreading scheme. Consider a random realization $\bar{\mathbf{S}}$, in which the adversaries observe a set of first-received messages $S_v \subseteq \mathcal{X}$ from each node $v \in V$. Now, supposing in these observations there exists a subset of t server nodes $\{v_1, v_2, \ldots, v_t\}$ whose own messages are included in the respective forwarded sets, i.e., $X_{v_i} \in \bar{S}_{v_i} \forall i = 1, 2, \ldots, t$. The macro-averaged precision in this case is

$$D_{\texttt{FS}} = \frac{1}{\tilde{n}} \sum_{i=1}^{t} \frac{1}{|\bar{S}_{v_i}|} \geq \frac{t^2}{\tilde{n} \sum_{i=1}^{\tilde{n}} |\bar{S}_{v_i}|} \geq \frac{t^2}{\tilde{n}^2}, \tag{8.10}$$

where the first inequality above is due to the arithmetic-mean harmonic-mean (A.M-H.M) inequality, and the second inequality is because the total number of messages is at most $\tilde{n}$. Equation (8.10) in turn implies that

$$\mathbb{E}[D_{\texttt{FS}} | T = t] \geq \frac{t^2}{\tilde{n}^2}. \tag{8.11}$$

The overall expected detection precision can then be bounded as

$$\begin{aligned}\mathbf{D}_{\texttt{FS}} = \mathbb{E}[D_{\texttt{FS}}] &= \sum_{t=0}^{\tilde{n}} \mathbb{P}(T=t)\mathbb{E}[D_{\texttt{FS}}|T=t] \\ &\geq \sum_{t=0}^{\tilde{n}} \mathbb{P}(T=t)\frac{t^2}{\tilde{n}^2} = \frac{\mathbb{E}[T^2]}{\tilde{n}^2} \geq \frac{\mathbb{E}[T]^2}{\tilde{n}^2} \\ &= \frac{\mathbb{E}[\sum_{v\in V_H} \mathbf{1}_{X_v \in \bar{S}_v}]^2}{\tilde{n}^2} \geq \frac{(p\tilde{n})^2}{\tilde{n}^2} = p^2, \end{aligned} \tag{8.12}$$

where the inequality in Equation (8.12) follows from Lemma 8.2.1. Finally by definition we have $\mathbf{D}_{\texttt{OPT}} \geq \mathbf{D}_{\texttt{FS}}$ and hence the theorem follows.

8.2.5.2 Proof of Theorem 8

We separate the proof into two parts: precision and recall. Both parts utilize the following result from [34], which is replicated without proof:

Lemma 8.2.2 (Optimal Estimators, [34])**.** *The precision-optimizing estimator for an adversary with observations A, is achieved by a matching over the bipartite graph $(V, \mathcal{X})$. Moreover, such a matching is a maximum-weight matching for edge weights $\mathbb{P}(X_v = x|A)$ on each edge $(v, x) \in V \times \mathcal{X}$ of the graph. The recall-optimizing estimator is a mapping that assigns each transaction $x \in \mathcal{X}$ to any server $v^* \in \arg\max_{v\in V} \mathbb{P}(X_v = x|A)$.*

Recall. We first show that the first-spy estimator is recall-optimal for Dandelion spreading, then that the first-spy estimator has an expected recall of p.

To show the first step, i.e., $\mathbf{R}_{\texttt{OPT}} = \mathbf{R}_{\texttt{FS}}$, Lemma 8.2.2 implies that we must show that for every message x, its exit node z (i.e., the node implicated by the first-spy estimator) maximizes $\mathbb{P}(X_v = x|A)$. For any message X_u, let $\Pi_u = (\Pi_{1,u}, \Pi_{2,u}, \ldots, \Pi_{L_u,u})$ be the path taken by a message from its source u $(= \Pi_{1,u})$ until it reaches an adversarial node $\Pi_{L_u,u}$ for the first time (L_u denotes the length of the path). From the adversary's observation $\mathbf{S}$, $\Pi_{L_u-1,u}$ and $\Pi_{L_u,u}$ are fixed as the exit node z and the first spy for X_u, respectively. Due to the specification of Dandelion spreading, the likelihood of this path, $\mathcal{L}(\Pi_u)$, is $\mathcal{L}(\Pi_u) = \prod_{i=1}^{L_u-1} \frac{1}{\deg(\Pi_{i,u})}$, where $\deg(v)$ denotes the out-degree of v. Assuming a uniform prior over candidate sources, we have $\mathbb{P}(X_v = x|A) \propto \mathcal{L}(\Pi_v)$. Since each node is assumed to have an out-degree of at least 1, this likelihood is maximized by taking the shortest path possible. That is, the maximum-likelihood path over all paths originating at honest candidate sources gives $z \in \arg\max_{v\in V_H} \mathbb{P}(X_v = x|A)$. Hence the first-spy estimator is also a maximum-recall estimator.

Now we analyze the recall of the first-spy estimator. Let $\mathcal{P}_v$ denote the event that v's parent (i.e., the next node in the line) is adversarial. Then the expected recall averaged over the set of honest nodes V_H is

$$\begin{aligned}\mathbf{R}_{\texttt{OPT}} = \mathbb{E}[R_{\texttt{FS}}|\mathbf{S}, G] &= \frac{1}{\tilde{n}}\mathbb{E}\left[\sum_{v\in V_H} \mathbb{1}\{\mathcal{P}_v\}\right] \\ \Rightarrow \mathbf{R}_{\texttt{OPT}} = \frac{1}{\tilde{n}}\sum_{v\in V_H} \mathbb{P}(\mathcal{P}_v) &= \frac{1}{\tilde{n}}\sum_{v\in V_H}\left(\frac{np}{n-1}\right) = p + O\left(\frac{1}{n}\right),\end{aligned}$$

where $\tilde{n} := |V_H|$ denotes the number of honest nodes.

Precision. The proof works by evaluating the cost incurred under various possibilities for the local neighborhood structure around a node in the network. For any honest server node $v \in V_H$, let $\mathcal{E}_v(i,j)$ denote the event that (i) i nodes preceding v are honest nodes, the $(i+1)$-th node preceding v is adversarial and (ii) j nodes succeeding v are honest nodes and the $(j+1)$-th node following v is adversarial. Also for ease of notation let $\mathcal{I}_v$ denote the event $\cup_{i>0,j>0}\mathcal{E}_v(i,j)$. Then the following lemmas hold true.

Lemma 8.2.3. *On a line-graph, for any $i, j > 0$, we have*

$$\begin{aligned}
\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(i,0))|\mathcal{E}_v(i,0)] &\leq \frac{1}{i+1} \\
\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(0,j))|\mathcal{E}_v(0,j)] &\leq \frac{1}{j+1} \\
\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(0,0))|\mathcal{E}_v(0,0)] &\leq 1 \\
\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{I}_v)|\mathcal{I}_v] &\leq \frac{1}{n(1-3p)}.
\end{aligned} \tag{8.13}$$

The proof can be found in Section 8.2.5.3.

Lemma 8.2.4. *On a line-graph, for $i, j > 0$ we have*

$$\mathbb{P}(\mathcal{E}_v(i,0)) \leq \left(p + \frac{1}{n}\right)^2 \left(1 - p + \frac{2}{n}\right)^i \tag{8.14}$$

$$\mathbb{P}(\mathcal{E}_v(0,j)) \leq \left(p + \frac{1}{n}\right)^2 \left(1 - p + \frac{2}{n}\right)^j \tag{8.15}$$

$$\mathbb{P}(\mathcal{E}_v(0,0)) \leq (p + 1/n)^2 \tag{8.16}$$

$$\mathbb{P}(\mathcal{I}_v) \leq (1-p)^2. \tag{8.17}$$

The proof can be found in Section 8.2.5.4.

Lemma 8.2.5. *If $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_k$ is a set of mutually exclusive and exhaustive events, and $v \in V_H$ is any honest server node, then*

$$\mathbf{D}_{\texttt{OPT}}(v) \leq \sum_{i=1}^{k} \mathbb{P}(\mathcal{E}_i)\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_i)|\mathcal{E}_i]. \tag{8.18}$$

The proof can be found in [34]. To complete the proof of Theorem 8, let use Lemma 8.2.5 with $\mathcal{E}_v(i,0), \mathcal{E}_v(0,j), \mathcal{E}_v(0,0)$ and $\mathcal{E}_v$ for $i, j > 0$ as the set of mutually exclusive and exhaustive events. Then the expected payoff at v can be bounded as

$$\begin{aligned}
\mathbf{D}_{\texttt{OPT}}(v) \leq &\sum_{i>0} \mathbb{P}(\mathcal{E}_v(i,0))\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(i,0))|\mathcal{E}_v(i,0)] \\
&+ \sum_{j>0} \mathbb{P}(\mathcal{E}_v(0,j))\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(0,j))|\mathcal{E}_v(0,j)] \\
&+ \mathbb{P}(\mathcal{E}_v(0,0))\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{E}_v(0,0))|\mathcal{E}_v(0,0)] \\
&+ \mathbb{P}(\mathcal{I}_v)\mathbb{E}[\max_{x\in\mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{I}_v)|\mathcal{I}_v],
\end{aligned} \tag{8.19}$$

where the values of the individual expectation and probability terms in the above Equation (8.19) have been computed in Lemmas 8.2.3 and 8.2.4 respectively. Using those bounds, we get

$$\begin{aligned}\mathbf{D}_{\mathtt{OPT}}(v) \leq & \sum_{i>0}\left(p+\frac{1}{n}\right)^2\left(1-p+\frac{2}{n}\right)^i\frac{1}{i+1}+\sum_{j>0}\left(p+\frac{1}{n}\right)^2\left(1-p+\frac{2}{n}\right)^j\frac{1}{i+1}\\ &+(p+1/n)^2+(1-p)^2\frac{1}{n(1-3p)}\\ \leq & \frac{2(p+\frac{1}{n})^2}{(1-p+\frac{2}{n})}\log\left(\frac{1}{p-\frac{2}{n}}\right)+\frac{(1-p)^2}{n(1-3p)}\\ \leq & \frac{2p^2}{1-p}\log\left(\frac{2}{p}\right)+O\left(\frac{1}{n}\right).\end{aligned} \tag{8.20}$$

Finally averaging the expected payoff $\mathbf{D}_{\mathtt{OPT}}(v)$ over each of the $\tilde{n}$ honest server nodes $v \in V_H$, we get the desired result.

8.2.5.3 Proof of Lemma 8.2.3

Consider a realization G of the network topology such that our desired event $\mathcal{E}_v(i,0)$ happens. In such a graph G, the node succeeding v is an adversarial node and the i nodes preceding v are honest. Let us denote this set of $i+1$ nodes – comprising of the i nodes preceding v and v itself – as W_v (i.e., the ward of v). Now, if the messages assigned to the nodes outside of W_v is denoted by $X(V_H \backslash W_v)$, then for any $x \in S_v$ we have $\mathbb{P}(X_v = x | G, \mathbf{S}, A, \mathcal{E}_v(i,0), X(V_H \backslash W_v)$

$$\begin{aligned}&= \frac{\mathbb{P}(X_v = x, \mathbf{S}, X(V_H \backslash W_v) | G, A, \mathcal{E}_v(i,0))}{\sum_{x \in S_v}\mathbb{P}(X_v = x, \mathbf{S}, X(V_H \backslash W_v) | G, A, \mathcal{E}_v(i,0))}\\ &= \frac{\mathbb{P}(X_v = x, X(V_H \backslash W_v) | G, A, \mathcal{E}_v(i,0))}{\sum_{x \in S_v}\mathbb{P}(X_v = x, X(V_H \backslash W_v) | G, A, \mathcal{E}_v(i,0))}\\ &= \frac{1}{i+1},\end{aligned} \tag{8.21}$$

by using the fact that the allocation of messages $\mathbf{X}$ is independent of the graph structure $(G, A, \mathcal{E}_v(i,0))$ and $\mathbb{P}(\mathbf{S} | X_v = x, X(V_H \backslash W_v), G, A, \mathcal{E}_v(i,0)) = 1$ on a line-graph. Now, taking expectation on both sides of Equation (8.21) we get

$$\begin{aligned}\mathbb{P}(X_v = x | \mathbf{S}, A, \mathcal{E}_v(i,0)) &= \frac{1}{i+1}\ \forall x \in S_v\\ \Rightarrow \max_{x \in \mathcal{X}}\mathbb{P}(X_v = x | \mathbf{S}, A, \mathcal{E}_v(i,0)) &= \frac{1}{i+1}\ \text{or}\\ \mathbb{E}[\max_{x \in \mathcal{X}}\mathbb{P}(X_v = x | \mathbf{S}, A, \mathcal{E}_v(i,0)) | \mathcal{E}_v(i,0)] &= \frac{1}{i+1}.\end{aligned} \tag{8.22}$$

By a similar argument as above, we can also show that

$$\begin{aligned}\mathbb{E}[\max_{x \in \mathcal{X}}\mathbb{P}(X_v = x | \mathbf{S}, A, \mathcal{E}_v(0,j)) | \mathcal{E}_v(0,j)] &= \frac{1}{j+1},\\ \mathbb{E}[\max_{x \in \mathcal{X}}\mathbb{P}(X_v = x | \mathbf{S}, A, \mathcal{E}_v(0,0)) | \mathcal{E}_v(0,0)] &= 1.\end{aligned} \tag{8.23}$$

Finally let us consider the case where v is an interior node, i.e., event $\mathcal{I}_v$ happens. As before, for a head-node u (an honest node whose successor is an adversarial node) let W_u denote the ward

containing u. Notice that under observations $\mathbf{S}, \Gamma(V_A)$ the adversaries know (i) the head and tail nodes of each ward (from $\Gamma(V_A)$) and (ii) the size of each ward ($|W_u| = |S_u|$). Therefore if a message x is such that $x \in S_u$ for some u, then

$$\begin{aligned}\mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{I}_v) &= \mathbb{P}(X_v = x, v \in W_u|\mathbf{S}, A, \mathcal{I}_v) \\ &= \mathbb{P}(v \in W_u|\mathbf{S}, A, \mathcal{I}_v)\mathbb{P}(X_v = x|v \in W_u, \mathbf{S}, A, \mathcal{I}_v) \\ &= \frac{|W_u| - 2}{|I|}\frac{1}{|W_u|} \leq \frac{1}{|I|} \leq \frac{1}{n(1-3p)}, \end{aligned} \tag{8.24}$$

where I denotes the set of all interior nodes and $|I| \geq n(1-3p)$ since each adversary is a neighbor to at most 2 honest server nodes. Hence we have

$$\mathbb{E}[\max_{x \in \mathcal{X}} \mathbb{P}(X_v = x|\mathbf{S}, A, \mathcal{I}_v)|\mathcal{I}_v] \leq \frac{1}{n(1-3p)}, \tag{8.25}$$

concluding the proof.

8.2.5.4 Proof of Lemma 8.2.4

First let us consider the event $\mathcal{E}_v(i, 0)$ in which node v has an adversarial successor, i honest predecessor nodes and an adversarial $i+1$-th predecessor. Let Y_v denote the position of node v in the line graph. Then

$$\mathbb{P}(\mathcal{E}_v(i, 0)) = \sum_{j=i+1}^{n} \mathbb{P}(Y_v = j)\mathbb{P}(\mathcal{E}_v(i, 0)|Y_v = j), \tag{8.26}$$

since v needs to be at a position on the line graph where at least $i+1$ predecessors are feasible. Now, for $i + 1 \leq j \leq n$, by a simple counting argument we have

$$\begin{aligned}\mathbb{P}(\mathcal{E}_v(i, 0)|Y_v = j) &= \left(\frac{np}{n-1}\right)\left(\frac{np-1}{n-2}\right)\left(\frac{\tilde{n}-1}{n-3}\right)\left(\frac{\tilde{n}-2}{n-4}\right)\cdots\left(\frac{\tilde{n}-i}{n-i-2}\right) \\ &\leq \left(p + \frac{1}{n}\right)^2\left(1 - p + \frac{2}{n}\right)^i\end{aligned}$$

Combining the above inequality with Equation (8.26) we conclude that

$$\mathbb{P}(\mathcal{E}_v(i, 0)) \leq \left(p + \frac{1}{n}\right)^2\left(1 - p + \frac{2}{n}\right)^i \tag{8.27}$$

for $i > 0$. By essentially a similar counting as above, we can also obtain the remaining Equations (8.15), (8.16) and (8.17) from the Lemma.

8.3 Acknowledgement

The authors thank S. Bakshi, S. Bhargava, B. Denby, and S. Venkatakrishnan for collaborations on the design and analysis of Dandelion. This chapter includes material from [34, 70].

Bibliography

[1] Amp: Atomic multi-path payments over lightning. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

[2] Bitcoin historical fee chart. https://bitinfocharts.com/comparison/bitcoin-median_transaction_fee.html.

[3] Ethereum Wiki proof of stake faqs: Grinding attacks. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs.

[4] Falcon network. https://www.falcon-net.org/.

[5] Internet service provider (isp) topology zoo. http://www.topology-zoo.org/.

[6] Kovri. https://getkovri.org/.

[7] Onion routed micropayments for the lightning network. https://github.com/lightningnetwork/lightning-onion.

[8] Particl. https://particl.com/.

[9] Qtum. https://qtum.com/.

[10] Raiden network. https://raiden.network/.

[11] Ripplenet. https://ripple.com/.

[12] Speedymurmurs software. https://crysp.uwaterloo.ca/software/speedymurmurs/.

[13] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-scalable byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[14] Mohammed Amin Abdullah and Moez Draief. Global majority consensus by local majority polling on graphs of a given degree sequence. *Discrete Applied Mathematics*, 180:1–10, 2015.

[15] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. *CoRR, abs/1612.02916*, 2016.

[16] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[17] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[18] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 435–446, New York, NY, USA, 2013. ACM.

[19] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[20] Gavin Andresen. Weak block thoughts... bitcoin-dev. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011157.html`.

[21] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[22] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007.

[23] Vivek Bagaria, Giulia Fanti, Sreeram Kannan, David Tse, and Pramod Viswanath. Prism++: a throughput-latency-security-incentive optimal proof of stake blockchain algorithm. In *Working paper*, 2018.

[24] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Deconstructing the blockchain to approach physical limits. `https://arxiv.org/abs/1810.08092`.

[25] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*, 2018.

[26] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[27] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptol. ePrint Arch., Tech. Rep*, 46:2018, 2018.

[28] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptol. ePrint Arch., Tech. Rep*, 46:2018, 2018.

[29] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 287–304. IEEE, 2015.

[30] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796, 2014.

[31] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29. ACM, 2014.

[32] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*, pages 122–134. IEEE, 2015.

[33] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[34] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):22, 2017.

[35] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multiparty computation for zk-snark parameters in the random beacon model. Technical report, Cryptology ePrint Archive, Report 2017/1050, 2017.

[36] Vitalik Buterin. On slow and fast block times, 2015. `https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/`.

[37] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[38] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical report/Dept. of Computer Science, ETH Zürich*, 260, 1997.

[39] Cardano. Cardano settlement layer documentation. `https://cardanodocs.com/technical/`.

[40] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167. ACM, 2016.

[41] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[42] Miguel Castro, Barbara Liskov, and et. al. Practical byzantine fault tolerance. In Paul J. Leach and Margo Seltzer, editors, *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, volume 99, pages 173–186, February 1999.

[43] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1825–1842. ACM, 2017.

[44] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1), 1988.

[45] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[46] David D Clark, Van Jacobson, John Romkey, and Howard Salwen. An analysis of tcp processing overhead. *IEEE Communications magazine*, 27(6):23–29, 1989.

[47] Thomas Clausen and Philippe Jacquet. Optimized link state routing protocol (olsr). Technical report, 2003.

[48] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *CCS*. ACM, 2010.

[49] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 177–190. USENIX Association, 2006.

[50] James Cruise and Ayalvadi Ganesh. Probabilistic consensus via polling and majority rules. *Queueing Systems*, 78(2):99–120, 2014.

[51] Leigh Cuen. 100 merchants can now trial Bitcoin's lightning network risk free. July 2018.

[52] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. Technical report, Technical report, 2016.

[53] George Danezis, Claudia Diaz, Emilia Käsper, and Carmela Troncoso. The wisdom of crowds: attacks and optimal constructions. In *European Symposium on Research in Computer Security*, pages 406–423. Springer, 2009.

[54] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie. Drac: An architecture for anonymous low-volume communications. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 202–219. Springer, 2010.

[55] Alex de Vries. Bitcoin's growing energy problem. *Joule*, 2(5):801–805, 2018.

[56] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013.

[57] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.

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

[59] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[60] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[61] Jacob Donnelly. What is the 'halving'? a primer to bitcoin's big mining change. 2016.

[62] Atilla Eryilmaz and R Srikant. Joint congestion control, routing, and mac for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1514–1524, 2006.

[63] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

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

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

[66] G. Fanti, P. Kairouz, S. Oh, and P. Viswanath. Spy vs. spy: Rumor source obfuscation. In *SIGMETRICS Perform. Eval. Rev.*, volume 43, pages 271–284, 2015.

[67] Giulia Fanti, Jiantao Jiao, Ashok Makkuva, Sewoong Oh, Ranvir Rana, and Pramod Viswanath. Barracuda: The power of l-polling in proof-of-stake blockchains. *arXiv preprint*, 2018.

[68] Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of wealth in proof-of-stake cryptocurrencies. *arXiv preprint arXiv:1809.07468*, 2018.

[69] Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of wealth in proof-of-stake cryptocurrencies. *arXiv preprint https://arxiv.org/pdf/1809.07468.pdf*, 2018.

[70] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):29, 2018.

[71] Giulia Fanti and Pramod Viswanath. Deanonymization in the bitcoin p2p network. In *Advances in Neural Information Processing Systems*, pages 1364–1373, 2017.

[72] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1(2):77–94, 1988.

[73] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

[74] V. Fioriti and M. Chinnici. Predicting the sources of an outbreak with a spectral technique. *arXiv:1211.2333*, 2012.

[75] MJ Fischer, N Lynch, and MS Paterson. Impossibility of distributed consensus with one faulty process, 1985.

[76] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[77] Cédric Fournet, Markulf Kohlweiss, George Danezis, Zhengqin Luo, et al. Zql: A compiler for privacy-preserving data processing. In *USENIX Security Symposium*, pages 163–178, 2013.

[78] M.J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. CCS*. ACM, 2002.

[79] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[80] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

[81] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083, 2016.

[82] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[83] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, 2003.

[84] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[85] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[86] Shafi Goldwasser. How to play any mental game, or a completeness theorem for protocols with an honest majority. *Proc. the Nineteenth Annual ACM STOC'87*, pages 218–229, 1987.

[87] Shafi Goldwasser. Multi party computations: past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 1–6. ACM, 1997.

[88] P. Golle and A. Juels. Dining cryptographers revisited. In *Advances in Cryptology-Eurocrypt 2004*, 2004.

[89] Aleksi Grym et al. The great illusion of digital currencies. 2018.

[90] James Hendricks, Gregory R Ganger, and Michael K Reiter. Low-overhead byzantine fault-tolerant storage. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 73–86. ACM, 2007.

[91] Alyssa Hertig. Lightning: The Bitcoin scaling tech you really should know. December 2017.

[92] Alyssa Hertig. Bitcoin fees are down big: Why it happened and what it means. February 2018.

[93] Chi-Yao Hong, Matthew Caesar, and P Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 127–138. ACM, 2012.

[94] Gur Huberman, Jacob D Leshno, and Ciamac C Moallemi. Monopoly without a monopolist: An economic analysis of the bitcoin payment system. 2017.

[95] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM computer communication review*, volume 18, pages 314–329. ACM, 1988.

[96] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 565–574. IEEE, 2000.

[97] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM computer communication review*, 32(4):89–102, 2002.

[98] Frank Kelly and Thomas Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):5–12, 2005.

[99] Justin Khim and Po-Ling Loh. Confidence sets for the source of a diffusion in regular trees. *arXiv preprint arXiv:1510.05461*, 2015.

[100] Aggelos Kiayias, Elias Koutsoupias, Mario Larangeira, Lars Brunjes, Dimitris Karakostas, and Aikaterini Stouka. Incentives and staking in cardano. `https://staking.cardano.org/`.

[101] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[102] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloxroute: A scalable trustless blockchain distribution network whitepaper.

[103] Leonid Kogan, Max Orhstrand, Giulia Fanti, and Pramod Viswanath. Economics of proof-of-stake payment systems. *Working paper*, 2018.

[104] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.

[105] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.

[106] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[107] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014.

[108] J Kubiatowicz. An architecture for global-scale persistent store. *Proc. ASPLOS'2000, Cambridge, MA, November*, 2000.

[109] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

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

[111] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 919–927. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[112] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[113] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[114] Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.

[115] Songzi Li, Mingchao Yu, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously. *arXiv preprint arXiv:1709.05748*, 2018.

[116] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 297–315. Springer, 2017.

[117] A. Y. Lokhov, M. Mézard, H. Ohta, and L. Zdeborová. Inferring the origin of an epidemic with dynamic message-passing algorithm. *arXiv preprint arXiv:1303.5315*, 2013.

[118] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[119] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. *IACR Cryptology ePrint Archive*, 2016:1054, 2016.

[120] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.

[121] John Medley. Bitcoin lightning network: Scaling cryptocurrencies for mainstream use. July 2018.

[122] Sarah Meiklejohn, C Christopher Erway, Alptekin Küpçü, Theodora Hinkle, and Anna Lysyanskaya. Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, volume 10, pages 193–206, 2010.

[123] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[124] Silvio Micali. Algorand: the efficient and democratic ledger. *CoRR, abs/1607.01341*, 2016.

[125] Chris Milling, Constantine Caramanis, Shie Mannor, and Sanjay Shakkottai. Network forensics: random infection vs spreading epidemic. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):223–234, 2012.

[126] Prateek Mittal, Matthew Wright, and Nikita Borisov. Pisces: Anonymous communication using social networks. *arXiv preprint arXiv:1208.6326*, 2012.

[127] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[128] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks. In *Network and Distributed Security Symposium*, 2015.

[129] Malte Möser and Rainer Böhme. Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *International Conference on Financial Cryptography and Data Security*, pages 19–33. Springer, 2015.

[130] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[131] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Security and Privacy, Symposium on*, pages 173–187. IEEE, 2009.

[132] Christopher Natoli and Vincent Gramoli. The balance attack against proof-of-work blockchains: The r3 testbed as an example. *arXiv preprint arXiv:1612.09426*, 2016.

[133] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 305–320. IEEE, 2016.

[134] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future internet*, 5(2):237–250, 2013.

[135] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

[136] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[137] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 315–324. ACM, 2017.

[138] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[139] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[140] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[141] Robin Pemantle. A time-dependent version of pólya's urn. *Journal of Theoretical Probability*, 3(4):627–637, 1990.

[142] P. C. Pinto, P. Thiran, and M. Vetterli. Locating the source of diffusion in large-scale networks. *Physical review letters*, 109(6):068702, 2012.

[143] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5*, 9:14, 2016.

[144] B. A. Prakash, J. Vreeken, and C. Faloutsos. Spotting culprits in epidemics: How many and which ones? In *ICDM*, volume 12, pages 11–20, 2012.

[145] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper (bitfury. com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016. pdf)*, 2016.

[146] Dongyu Qiu and Rayadurgam Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *ACM SIGCOMM computer communication review*, volume 34, pages 367–378. ACM, 2004.

[147] Michael K Reiter, Matthew K Franklin, John B Lacy, and Rebecca N Wright. The ω key management service. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 38–47. ACM, 1996.

[148] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

[149] Peter R Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016.

[150] Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies,?, 2018.

[151] Rodrigo Rodrigues, Miguel Castro, and Barbara Liskov. Base: Using abstraction to improve fault tolerance. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 15–28. ACM, 2001.

[152] Stefanie Roos, Martin Beck, and Thorsten Strufe. Anonymous addresses for efficient and resilient routing in f2f overlays. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[153] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint https://arxiv.org/pdf/1809.10361*, 2017.

[154] Ron Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[155] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In *Annual International Cryptology Conference*, pages 555–572. Springer, 1999.

[156] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.

[157] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[158] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[159] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[160] D. Shah and T. Zaman. Detecting sources of computer viruses in networks: theory and experiment. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 203–214. ACM, 2010.

[161] D. Shah and T. Zaman. Rumor centrality: a universal source detector. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 199–210. ACM, 2012.

[162] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[163] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. *Journal of Computer Security*, 13(6):839–876, 2005.

[164] Hyun Song Shin. Cryptocurrencies and the economics of money, Jun 2018.

[165] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. Routing cryptocurrency with the spider network. In *Proceedings of the ACM Hotnets 2018*.

[166] Y Sompolinsky and A Zohar. Phantom: A scalable blockdag protocol, 2018.

[167] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[168] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[169] Rayadurgam Srikant and Lei Ying. *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[170] Statoshi. Bandwidth usage. `https://statoshi.info/dashboard/db/bandwidth-usage`.

[171] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 526–545. IEEE, 2016.

[172] TierNolan. Decoupling transactions and pow. Bitcoin Forum. `https://bitcointalk.org/index.php?topic=179598.0`.

[173] Kyle Torpey. Greg maxwell: Lightning network better than sidechains for scaling bitcoin, 2016. `https://bitcoinmagazine.com/articles/greg-maxwell-lightning-network-better-than-sidechains-for-scaling-bitcoin-1461077424/`.

[174] Paul F Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 35–42. ACM, 1988.

[175] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 115–126, New York, NY, USA, 2012. ACM.

[176] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Scalable private messaging resistant to traffic analysis.

[177] Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.

[178] Jan Vermeulen. Bitcoin and ethereum vs visa and paypal ? transactions per second. April 2017.

[179] Visa. Visa acceptance for retailers. `https://usa.visa.com/run-your-business/small-business-tools/retail.html`.

[180] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[181] Dan S Wallach, Peter Druschel, et al. Enforcing fair sharing of peer-to-peer resources. In *International Workshop on Peer-to-Peer Systems*, pages 149–159. Springer, 2003.

[182] Z. Wang, W. Dong, W. Zhang, and C.W. Tan. Rumor source detection with multiple observations: Fundamental limits and algorithms. In *ACM SIGMETRICS*, 2014.

[183] Ethereum Wiki. Sharding roadmap. `https://github.com/ethereum/wiki/wiki/Sharding-roadmap`.

[184] Herbert S Wilf. *Algorithms and complexity*. AK Peters/CRC Press, 2002.

[185] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–182, 2012.

[186] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[187] Qian Yu, Netanel Raviv, Jinhyun So, and A Salman Avestimehr. Lagrange coded computing: Optimal design for resiliency, security and privacy. *arXiv preprint arXiv:1806.00939*, 2018.

[188] M. Zamani, J. Saia, M. Movahedi, and J. Khoury. Towards provably-secure scalable anonymous broadcast. In *USENIX FOCI*, 2013.

[189] Bassam Zantout and Ramzi Haraty. I2p data communication system. In *Proceedings of ICN*, pages 401–409. Citeseer, 2011.

[190] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 863–880. IEEE, 2017.

[191] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Coca: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)*, 20(4):329–368, 2002.

[192] K. Zhu and L. Ying. A robust information source estimator with sparse observations. *arXiv preprint arXiv:1309.4846*, 2013.

[193] Yi Ming Zou. Representing boolean functions using polynomials: more can offer less. In *International Symposium on Neural Networks*, pages 290–296. Springer, 2011.