A dissertation

submitted in partial fulfillment of the

requirements for the degree of

University of Washington

Reading Committee:

Program Authorized to Offer Degree:

University of Washington

Abstract

Fair Ordering in Blockchains

Soubhik Deb

Chair of the Supervisory Committee:

Sreeram Kannan

Department of Electrical & Computer Engineering

Blockchains are an emerging paradigm for permissionless distributed computing, that can remove digital intermediaries while at the same time providing trusted computing and storage guarantees. While applications of blockchains beyond cryptocurrencies into generalized digital platforms, are still in their infancy, blockchain technology is expected to create several trillion dollars of value in the next few years, potentially aiding the democratization of trust, data and control in digital platforms. Given the increasing volume of value being exchanged on blockchains, transaction order manipulation attacks have become extremely common in public blockchains such as Ethereum, costing hundreds of millions of dollars each year. In these blockchains, a miner can unilaterally determine the order of transactions inside a block, and this ordering is not checked by other users, leaving room for the miner to manipulate the order for its own benefit. Hence, coming up with correct primitives and protocols to achieve those primitives is of paramount importance. Furthermore, even if the miner is honest and includes the transactions in a fair manner, it can be subjected to frontrunning by other users as propagation of transaction takes some time during which the first transaction can be observed and can be frontrunned. This necessitates fast peer-to-peer propagation. In this thesis, we address the various challenges that are being faced from the point of manipulation of transaction order. This required extensive research across the blockchain stack starting from modifying how messages are communicated across peer-to-peer network to building adapters on byzantine fault tolerant consensus protocols.

I would like to dedicate this thesis to everyone who has helped me to climb till here.

Acknowledgement

I would like to begin with a big thank you to my advisor Sreeram Kannan. He has been more than an advisor and has, over the years, given me good innumerable good advices on navigating life. A true mentor and like a big brother to the end indeed. Obviously, his clarity on how blockchains should work has helped me a lot in clarifying my understanding also. I always enjoyed our late night discussions, especially, during COVID times. Thanks SK! I am looking forward to working together in more awesome projects in future.

My parents, Sitangshu Deb and Ima Deb, have played a super big role in my upbringing. I am thankful to them for their emphasis and sacrifices for the sake of my education. They always have tried their best to encourage me to be curious, have a growth mindset and be fearless in trying out new things. Thank you Baba and Maa! My little brother Rishob has always been a constant friend since childhood. Fiercely loyal in the past and even now. Thanks brother being there with me. Being far from home, I miss y'all a lot.

Next, I would like to acknowledge my best friend, my life partner, my love, Madhuri Mahendra Nagare. Thanks for having my back whenever I am in a problematic scenario and for tolerating my mischiefs everyday. I have always found talking with you to have a huge calming effect on me. And I really appreciate you answering my questions on various Machine Learning-related topics and for playing devils advocate with my ideas.

Living in US has been made lot more comfortable in the company of the friends. I would like to give a shout-out to Bowen, Robert, Viswa, Jeffrey, Vyas, Daniel, Gautham, Siddhartha, Chris M, Sam Laf for daily bantering and for showing the ropes on how things are done in America.

Contents

List of Figures

List of Tables

Chapter 1

Perigee: Efficient Peer-to-Peer Network Design for Blockchains

A key performance metric in blockchains is the latency between when a transaction is broadcast and when it is confirmed (the so-called, confirmation latency). While improvements in consensus techniques can lead to lower confirmation latency, a fundamental lower bound on confirmation latency is the propagation latency of messages through the underlying peer-to-peer (p2p) network (in Bitcoin, the propagation latency is several tens of seconds). The *de facto* p2p protocol used by Bitcoin and other blockchains is based on random connectivity: each node connects to a random subset of nodes. The induced p2p network topology can be highly suboptimal since it neglects geographical distance, differences in bandwidth, hash-power and computational abilities across peers. We present Perigee, a decentralized algorithm that automatically learns an efficient p2p topology tuned to the aforementioned network heterogeneities, purely based on peers' interactions with their neighbors. Motivated by the literature on the multi-armed bandit problem, Perigee optimally balances the tradeoff between retaining connections to known well-connected neighbors, and exploring new connections to previously-unseen neighbors. Experimental evaluations show that Perigee reduces the latency to broadcast by 33%. Lastly Perigee is simple, computationally lightweight, adversary-resistant, and compatible with the selfish interests of peers, making it an attractive p2p protocol for blockchains.

1.1 Introduction

In 2008, Satoshi Nakamoto proposed Bitcoin as a decentralized currency system over a peer-to-peer (p2p) network, with the blockchain protocol as its underlying technology for maintaining a public ledger of payment transactions [128]. Since then, there has been a proliferation of applications leveraging the power of blockchains as a core component, for implementing cryptocurrencies, smart contracts, supply chain management, etc. [58]. Today, the combined market capitalization of all cryptocurrencies is around 280 billion dollars with a rapidly increasing user base [6].

A key problem facing blockchain systems today is scalability—for example, the Bitcoin network can currently support a maximum of only 10 transaction per second [155], compared to 1700 transactions per second on Visa. A blockchain protocol functions by periodically consolidating transactions and broadcasting them as "blocks" over the network. Recent works have constructed new consensus protocols to improve confirmation latency and throughput in both the permissioned [134, 21] as well as the permissionless settings [28, 94]. There have also been methods to compress [130] and code [54] blocks while forwarding. Despite these improvements, a fundamental factor limiting the performance of blockchain systems is the inherent message propagation delay introduced by the p2p network. A block experiences delays from various factors during propagation, such as due to link latencies and processing delays for verifying blocks at each peer. It is known that improving the propagation delay directly improves key performance metrics of the system: transaction throughput, latency in confirming transactions, and security [28].

Measurement studies over Bitcoin [60, 67] report that it takes on an average 79 seconds for a block to reach 90% of nodes in the network. Whereas the median round-trip-time between hosts on the Internet is <300ms [98], the median bandwidth of Bitcoin nodes is 33Mbps [60], and the average time taken to validate a block is <200ms [93]. Blocks have an average size of 1MB in Bitcoin today. These numbers show that the time it takes for a block to propagate to the majority of the network is 40× larger than the time it takes to verify and relay a block between two nodes (<1 or 2s). With an estimated number of less than 11,000 nodes in Bitcoin [3], and each node making connections to at least 8 other nodes [125], a key reason for the disproportionately large propagation delay today is due to the ineffective way in which nodes are connected to each other (*i.e.,*the topology) in the p2p network.

The design of p2p networks for efficient content storage and lookup has a long history [117, 156, 149]. However, blockchains require only a simple broadcast primitive (for example, unicast messages directed to a particular node or lookup for specific content are not supported) and this primitive needs to be robust to adversarial action. This has led to Bitcoin following a random connection policy, where each node chooses its neighbors randomly from among a set of known nodes. While the random graph topology is simple, robust and provides good connectivity (from a graph theoretic standpoint), it is oblivious to differences in round-trip-time latencies between different nodes, heterogeneity in node bandwidth and block verification times. This inherently worsens the overall delay experienced by broadcasted blocks; for example, a block is likely to make several back-and-forth trips across distant continents before reaching a node. In this paper, we consider the question of how to optimally connect nodes in the Bitcoin network (and blockchain p2p networks in general), in a way that is aware of link and node heterogeneities, so that the broadcast time of blocks is minimized.

We present Perigee a decentralized protocol that adaptively decides which neighbors a node should connect to, purely based on the node's past interactions with its neighbors. Our protocol is motivated by the classical multi-armed bandit problem [24]. Nodes in Perigee balance the trade-off between retaining old neighbors with good connectivity, and exploring new neighbors with potentially better connectivity. In Perigee, a node quantifies its interactions with its neighbors by looking at the block arrival times. Neighbors that consistently deliver blocks quickly are favored, while others are disconnected. Perigee also continuously forms connections to a small number of nodes randomly, for discovering previously unseen but well-connected nodes. Our approach of purely using block arrival times to select neighbors is automatically tuned to heterogeneity in link latencies, block validation delays and node bandwidth. The end result is a topology that is very tight: experimental results show Perigee improves overall propagation delay by 33% compared to the state of the art (§1.5).

Modifying the p2p topology for faster block propagation has been considered in prior works; e.g., in Kadcast [148] the authors propose a structured p2p overlay as a faster alternative to the random topology. However, such a structured topology is still oblivious to link latencies, block validation times and node bandwidth, which renders its performance to be only slightly better than the random topology (§1.5). One way to take link latency into account is by using the geographical location of nodes, inferred from their IP addresses, to select neighbors [19]. However, this approach does not accurately reflect propagation latencies since nodes frequently use proxy-servers, VPN and Tor to run nodes, not to mention potential geo-location spoofing attacks by adversaries. Even more importantly, this approach also remains oblivious to the differing processing power and bandwidth possessed by different nodes.

In contrast, Perigee does not use any explicit property about a node, and is thus much more robust to spoofing attacks. Another line of work proposes high-speed block distribution networks (e.g., BloXroute [109], Falcon [4], Fibre [5]) to reduce block propagation times. These solutions are not fully decentralized, as using them requires nodes to place trust on the relay network. The routes in these networks could be also susceptible to man-in-the-middle attack. Nevertheless, even if such relay networks are present, Perigee automatically adapts its topology to exploit those networks (§1.5.4).

Perigee naturally incentivizes nodes to follow protocol—if a node deviates from protocol (e.g., stops relaying blocks, or does not update its neighbors using Perigee), then its neighbors will penalize the node by disconnecting from it in the future. Consequently, the deviant node will lose out on receiving blocks in a timely manner.

Finally Perigee maintains a subset of random neighbors at all times, thus making it less susceptible to eclipse attacks.

1.1.1 Background

Blockchain applications use a distributed, replicated ledger—called the blockchain—for storing and updating, collective states of application's end-users. Bitcoin is a popular example of a blockchain application. In Bitcoin, the blockchain contains the sequence of all payment transactions made by users since the very beginning of Bitcoin. The public nature of these transaction logs allows a payee to unilaterally verify the authenticity of incoming payments, without relying on third party organizations. Thus it is a fully decentralized payment system, a property that has contributed significantly to its growth and popularity.

Bitcoin architecture

Bitcoin operates over a p2p network. In Bitcoin, when a user makes a payment transaction, first a transaction message specifying the sender, recipient, and payment amount is created by the user. The transaction is then broadcast to other peers over the network. As new transactions are propagated over the network, special peers called miners accumulate these transactions, verify their authenticity and consolidate them into individual transaction *blocks* in a process called mining. A block in Bitcoin can contain a few thousand transactions today. Miners compete for mining each block, as they receive a monetary reward (funded by transaction fees) for mining a block. To ensure immutability in the sequence of previously mined blocks, miners are obliged to include hash of the previous block and solve a computationally difficult cryptographic puzzle while mining. When a block is mined, the miner shares the block with the rest of the network by broadcasting it. A peer receiving a freshly mined block first verifies its authenticity, before appending the block to its local copy of the blockchain or relaying the block to other neighbors.

Block propagation delay and performance

Blocks are broadcast in Bitcoin via flooding; when a peer receives a new block, it announces the hash of the block to all its neighbors via an `INV` message. Subsequently, neighbors who have not yet received the block respond with a `GETDATA` message requesting for the block, and the block is relayed to them. The process repeats until all the peers in the network have received the new block.

The performance of Bitcoin, and other cryptocurrencies, is measured by their (i) *throughput*, which is the average rate at which transactions are confirmed in the blockchain per second, (ii) *confirmation latency*, the time taken such that the probability for removing an honest transaction from the blockchain becomes sufficiently small, and (iii) *security*, the extent of adversarial peers the network can tolerate before the blockchain loses its immutability property [28]. Cryptocurrencies today offer strong security guarantees, but are lacking in their throughput and confirmation latencies compared to mainstream payment systems. For example, Bitcoin promises its blockchain cannot be compromised as long as more than 50% of the miners are honest. However, compared to the average throughput of 1700 transactions per second in the Visa network, the average throughput in Bitcoin today is just 3–7 transactions per second, and the latency is 1 hour [60].

A key factor affecting the throughput, confirmation latency and security is the propagation delay of blocks. If the propagation delay is too large, then there is a higher probability of mining of a block while another block at the same blockchain height is being propagated across the network—a phenomenon called forking [67]—reducing network throughput. The confirmation latency is also physically lower bounded by the propagation delay of the underlying p2p network [28]. Furthermore, a large propagation delay can help an adversary to execute double spending and block-withholding attacks [151].

How the p2p topology impacts delay

The dynamics of block propagation in Bitcoin has been empirically observed to follow a pattern similar to randomized rumor spreading in networks [67]. For instance, when a block is mined and broadcast, it first spreads exponentially fast to peers that are close to the source, before slowing down exponentially and reaching the remaining peers [67, 101]. Prior works have extensively analyzed (both empirically, and theoretically) rumor spreading on different network topologies [71, 84], and have shown that rumors

spread substantially faster in certain topologies than others. Specifically, scale-free graphs spread rumors significantly faster (in sub-logarithmic time) than random graphs. Doerr et al. [71] report that on social networks (e.g., the Twitter topology), rumors spread even faster than on scale-free graphs.

While the rumor-spreading model is considerably simpler compared to the dynamics of block propagation in Bitcoin's network (e.g., it does not model heterogeneity in link latencies, or bandwidth) it illustrates the potential benefits of carefully designing the p2p topology. An optimal peer connection protocol should not only imbibe essential properties of a fast rumor-spreading network and take peer heterogeneity into account, but should also be implementable in a decentralized manner without introducing new vulnerabilities.

1.1.2 Problem Statement and Contributions

We consider re-designing Bitcoin's p2p topology, to minimize the time taken by blocks to propagate over the network. The topology is constructed using a fully decentralized protocol running at all the peers. A peer may choose to not follow protocol, or even act adversarially, but we assume there exist peers, whose aggregate compute power amounts to more than 50% of the total compute power in the network, that are honest [128]. Each honest peer seeks to connect to a set of neighboring peers, to minimize the time it takes for a block mined by the peer to reach a majority (e.g., 90%) of the compute power in the network. Honest peers are also interested in receiving blocks mined by a majority of other peers as early as possible. Our main contributions are as follows.

Fundamental bounds on delay. We present a theoretical model for analyzing block propagation delay in Bitcoin, that explicitly models heterogeneity in the communication latencies between peers. Our model is based on a line of work in the networking systems, which has proposed that latencies between hosts on the Internet can be accurately predicted by embedding the hosts on to a metric space [61]. With this model, we show that inter-connecting peers randomly leads to propagation delays that are logarithmically worse compared to the underlying point-to-point latencies between peers (§1.3.1). Conversely, we also show that a topology in which peers choose neighbors with whom they have a small round-trip-time latency, provides asymptotically the best possible propagation delay (§1.3.3).

Optimal algorithm. We propose Perigee, a decentralized neighbor-selection protocol, that adaptively decides which neighbors to connect to purely based on the interactions between a peer and its neighbors (§1.4). Perigee is motivated by the classical multi-armed bandit problem [24], in which an agent—faced with a decision to choose one among many options with a priori unknown rewards—adaptively tries the different options and zeros-in on the best choice. A core tenet of algorithms for solving the multi-armed bandit problem is balancing exploration (trying out a previously unexplored option) with exploitation (choosing an option that has already been tried before). In our case, each peer is an agent that is faced with choosing the best set of neighbors, among different choices for neighbors. Interpreting the p2p topology design problem as an instance of multi-armed bandit problem is a key novelty of our paper and, to our best knowledge, has not been proposed before. Further, our experiments show that the topology that is adaptively learned by Perigee has striking statistical similarities to the theoretically optimal topology (§1.3.3). In addition to minimizing propagation latency, Perigee is attractive also for the following reasons:

- It is lightweight.
- It is compatible with the self-interests of peers—each peer selfishly tries to select the best neighbors for itself.
- It supports incremental deployment: peers following Perigee would see improvements in how quickly they can send or receive blocks, compared to those that do not follow Perigee.
- It is robust against adversarial actions: a Perigee peer does not need to know much about a candidate neighbor (e.g., its geographical location, or the round-trip latency to the neighbor) to decide whether to connect to it.
- It incentivizes peers to relay blocks promptly.

- It is naturally adaptive to varying hash-power. Each node tries to optimize its distance from an average block source, rather than from an average node.

1.2 System Model

1.2.1 Network Model

We model Bitcoin's p2p network as an undirected graph $G(V,E)$, where V is the set of nodes, and E is the set of edges, or links, between the nodes. A node refers to a Bitcoin server (e.g., a miner), that can accept incoming TCP connection requests from other servers and clients. Clients on Bitcoin are end-devices that are not able to accept incoming TCP connection requests (e.g., because they are behind a NAT). Once a TCP connection has been established between two nodes, communication can happen in both directions. We focus in this work, on Bitcoin servers as they form the core of the p2p overlay—servers tend to be always on, and the time taken for a block to propagate is largely affected by the interconnection network between the servers. We focus on minimizing the latency of propagating blocks, not transactions, in this work (we define the objective formally in §1.2.2). It is well known that transaction throughput and confirmation latencies in Bitcoin are directly correlated with block propagation times [28]. Moreover, a p2p network that is optimized for rapidly broadcasting blocks would also minimize transaction broadcast time, as clients are likely to connect to well-connected server nodes (e.g., using [3]). However, our protocol is general, and can readily be adapted to optimize transaction propagation times as well.

For any two nodes $u,v \in V$, we assume the latency of sending a block from u to v or from v to u, via a TCP connection between u and v, is a constant $\delta_{(u,v)} \geq 0$. The latency here includes transmission delay, in-network (propagation, queueing etc.) delays and protocol-specific message exchange overheads (e.g., `inv`, `getdata` exchange in Bitcoin) while sending a block. $\delta_{(u,v)}$, for a pair of nodes u,v, depends on various factors: the size of each block, the Internet access bandwidths at u and v, the physical distance between the nodes, and the extent of congestion in the network. We assume these factors are slowly varying compared to the timescale of our algorithm. Each node $v \in V$ also spends a fixed amount of time Δ_v, for cryptographically verifying the authenticity of a block it receives. Δ_v varies between nodes depending on their processing power. The fraction of hash power a node v has, relative to the total hash power of the network, is denoted by f_v.

We assume blocks are periodically generated (e.g., once every 10 minutes) and broadcast over the network. The probability that a node v generates the block in a round is proportional to its hash power f_v. When a node u mines a block, or receives a block from a neighbor, it immediately starts relaying the block to each neighbor v, taking a time $\delta_{(u,v)}$ to finish relaying. For simplicity we also assume that the connection updates execute synchronously at all the nodes, immediately after a block is broadcast on the network.

At any time, each node maintains $d_{\text{out}} = 8$ outgoing connections, and has $d_{\text{in}} \leq 20$ incoming connections. In practice, Bitcoin nodes can have up to 8 outgoing and 125 incoming connections [125]. To discover peers in the network, Bitcoin nodes also maintain a local database called `addrMan`, which they regularly update by exchanging messages to neighbors. A bootstrapping server provides with a list of addresses for a freshly joining peer. However, we assume each node know the IP addresses of all other nodes.

1.2.2 Performance Metrics

For each $v \in V$, we compute the minimum overall delay λ_v it takes for a block mined and broadcast by v to reach nodes totalling to at least 90% of the hash power in the network. The objective for each $v \in V$ is to choose neighbors such that λ_v is minimized. By symmetry, this objective would equivalently also minimize the time taken by blocks mined by a majority of other nodes to reach v.

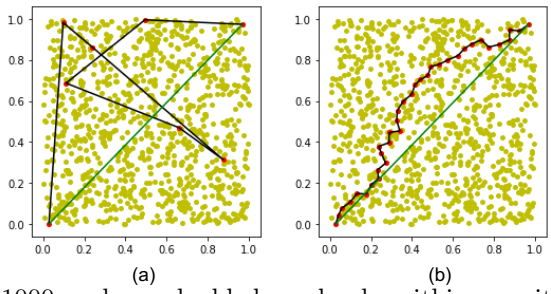


Figure 1-1: Example of 1000 nodes embedded randomly within a unit-square. (a) If nodes are interconnected according to a random topology, the shortest path between two points can be much longer than the Euclidean distance between the points. (b) If nodes are interconnected using a carefully designed topology (e.g., a geometric graph; see §1.3.3), significantly better paths, with length close to the Euclidean distance, are possible.

1.3 Baseline Algorithms

1.3.1 Random

The random connection policy is a simple algorithm that is widely deployed in many cryptocurrency systems today. In this algorithm, a node maintains a list of IP addresses of a small number of nodes that are currently active in the network. Initially a bootstrapping server provides the node with such a list; subsequently the list is updated (i.e., new addresses are added, while stale ones are removed) by gossiping any changes in the set of neighbors for each node, over the network. Intuitively, if connections are formed randomly on a world-wide network, then any path—and in particular the shortest path—between two nodes u,v would likely pass through intermediate nodes that are not located close to the shortest geographical route (*i.e.,*the geodesic) connecting u and v. Such less-than-direct paths would prolong the propagation delays of blocks sent on the network. Moreover, even with queueing delays on the Internet, we can show that a random topology leads to paths with latencies significantly larger than those of paths on optimal topologies. Based on extensive measurement studies, prior works [61] have empirically shown that endhosts on the Internet can be *embedded* on a high-dimensional metric space (e.g., $\mathbb{R}^5$) such that the metric distance between any two endhosts accurately predicts the communication latency between the hosts. However, the paths on a randomly connected network are unlikely to remain close to the geodesic shortest route between hosts, on the embedded high-dimensional space.

Example. To illustrate this, consider an example of a network embedded in the unit square $[0,1]\times[0,1]$, as shown in Figure 1-1. The green points within the square are drawn uniformly randomly and represent the nodes in the network. The Euclidean distance $||u-v||_2$ between any two nodes u,v is the one-way latency of sending a message (e.g., a transaction, or a block) from u to v or vice-versa.[1] Now, consider connecting each node in the unit-square randomly to 3 other nodes. Figure 1-1(a) shows the shortest path on this topology, between two nodes a and b that are closest to the bottom-left and top-right corner of the square. However, due to the meandering nature of paths in a random topology, the latency between a and b is much greater than the point-to-point latency $||a-b||_2$ between them. In contrast, a geometric graph topology (to be discussed shortly in §1.3.3) has a shortest path between a and b that is much closer to the geodesic shortest path (straight line between a and b), as shown in Figure 1-1(b). We formally show the suboptimality of the random topology next.

Suboptimality of the random algorithm. Let $[0,1]^d$ be the d-dimensional hypercube ($d \geq 2$), equipped with the Euclidean metric, and let $V = \{x_1, x_2, \ldots, x_n\}$ denote the nodes in the network. To model the point-to-point latencies between different pairs of nodes, *i.e.,*the latency between pairs of nodes if they are directly connected to each other, we consider an embedding of V on to $[0,1]^d$, in which each node x_i is mapped to a point X_i chosen uniformly randomly over $[0,1]^d$. The point-to-point latency between any two nodes x_i,$x_j \in V$, is then simply $||X_i - X_j||_2$.

Next, to model random connections between nodes, for each pair of nodes x_i,x_j we let x_i and x_j

[1] From the literature and results on metric-embedding of Internet hosts, we assume that message latency from u to v is equal to the latency from v to u.

have a link between them with probability p, independent of other links. Equivalently, we can consider each pair of points X_i,X_j on the embedded space to have a link between them with probability p, independent of other links. The resulting random graph of points $\{X_1,X_2,...,X_n\}$ is denoted by $\tilde{G}(\tilde{V},\tilde{E})$. The network latency dist(i,j) between any two points X_i,X_j is the time taken for a message broadcast by node i (resp. node j) to reach node j (resp. node i). This is computed as the total weight of edges on the shortest path between X_i and X_j on $\tilde{G}$, where the weight of each edge $(X_u,X_v) \in \tilde{E}$ is given by $||X_u - X_v||_2$. Clearly, the maximum point-to-point latency between any two points is bounded by $\sqrt{d}$, which is the Euclidean distance between the diagonal points [0,0,...,0] and [1,1,...,1] on the hypercube. However, due to the random nature of the graph, the typical network latency between any two nodes i,j can be a logarithmic factor worse as shown by the following Theorem.

Theorem 1 ([86]). *For any pair of nodes x_i,$x_j \in V$ and $p \leq c\log n/n$, where $c = c(n) = O(1)$, we have*

$$\text{dist}(i,j) \geq \frac{(\log n)^{1-\frac{1}{d}}}{8d^{3/2}e^d(\log\log n)^2 c^{1/d}} ||X_i - X_j||_2, \tag{1.1}$$

with probability $1 - o(1)$.

(Proof in Freize et al. [86].)

In Theorem 1 above, $p \leq c\log n/n$ connotes a small average degree of $c\log n$ per node in the network. The latency bound in Equation (1.1) holds asymptotically almost surely for any pair of nodes i,j because of our assumption that each node is embedded on to a random point in the hypercube.[2] In reality, while connection patterns across nodes can change randomly with time, the point-to-point latencies between nodes may not vary significantly. Nevertheless, for nodes that are not too close to each other, Theorem 1 suggests that the latency between them on a random network must be logarithmically worse.

1.3.2 Connecting Based on Geography

A key reason the randomly formed topology suffers from suboptimal path delays (Theorem 1), is due to a lack of sufficient connectivity between nodes that are in close proximity (*i.e.,*have small delay) to each other in the hypercube. If the size of the network is large, each node is likely to choose neighbors that are all far away, as the number of distant nodes is much greater than the number of nearby nodes. Therefore, even if a message reaches the general vicinity of node fast, it likely needs to spend a disproportionate amount of time to actually reach the node, due to the lack of any direct, low-delay paths. To ensure good connectivity in this "last mile", it is desirable for nodes to connect not only to nodes that are far away, but also those close by.

In practice, it is difficult for nodes to a priori know the round-trip-times to other nodes, without actually connecting to them first. However, recent work [33] has proposed using the geographical location of a node—which can be estimated based on it's IP address [99]—as a proxy for predicting whether the connection latency to the node is likely to be large or small. If the geographical locations of nodes are known, then a natural method to improve the random protocol, is to select a few neighbors among those that are geographically close, and then choose the rest of the neighbors randomly. For instance, if we cluster nodes according to the continents they are from, then a node located in North America can have four neighbors that are also in North America, and four other neighbors from other continents (e.g., Asia, Europe).

In our evaluations (§1.5), we show that the above protocol, does indeed perform better compared to the random protocol. However, the question remains whether this protocol can be improved even further. For instance, we have clustered nodes based on the continent in which they are located, but it is unclear if a different way of grouping nodes would have fared better. We also assign 50% of a node's connections to in-cluster nodes, and the remaining 50% to nodes outside the cluster. The optimal balance between the number of connections made within and outside of the cluster, is again unclear. In practice a node may be malicious and try to spoof it's true geographical location (e.g., via proxies, or VPN), which can also significantly degrade the utility of the algorithm. Lastly, the assumption that the geographic distance to a node dictates the latency to it is only a coarse approximation [98].

[2] This leads to any two points X_i,X_j being "well-separated" on the hypercube with high probability.

1.3.3 Theoretical Optimum

To understand how much better an optimal topology can be, we consider a geometric graph in which two nodes are connected if the latency between them is less than a threshold r. Compared to the random topology, in which neighbors are selected completely agnostic of their delay or geography, the geometric graph represents the other extreme where all neighbors are chosen to be within some small delay. Following the model for latency in which nodes are randomly embedded within a d-dimensional unit-hypercube (§1.3.1), we can show that the shortest path distance between any two nodes is at most a constant factor larger than their Euclidean distance.

Theorem 2 ([154, 85]). *For a geometric graph with threshold $r = \Theta((\log n/n)^{1/d})$, there exists a constant ξ such that for any two nodes x_i,x_j in the same connected component with $||X_i - X_j||_2 = \omega(r)$, it holds that* $\text{dist}(i,j)$ *is at most $\xi||X_i - X_j||_2$ with probability $1 - o(1)$.*

(Proof in Friedrich et al. [85])

The superior path delay of the geometric graph stems from nodes having a strong connectivity to other nodes in their local vicinity, which creates paths traversing closely to the geodesic between any two nodes (Figure 1-1(b)). We note that the geometric graph is not the only construction with order-optimal path delay—a recent line of work [68, 20] has proposed other efficient topology constructions also providing order-optimal path delays, for points embedded in a metric space. For example, in Chan et al. [53], the authors propose a decentralized algorithm for constructing a low-stretch spanner where the graph distance between any two nodes is at most a constant factor worse than their Euclidean distance. Their algorithm is also robust against node faults.

The metric embedding model for node latencies, discussed in §1.3.1, §1.3.2 and the present section, is useful as a simple, tractable theoretical model for analyzing competing topology constructions. While the model captures first-order differences in point-to-point latencies across nodes, blocks in the Bitcoin network also suffer from delays due to transmission (if the available bandwidth is small, relative to the block size), and block validation. Measurement studies on the Bitcoin network, report a wide skew in these delays across different nodes; e.g., in one study [60] conducted in 2015, the bandwidth of Bitcoin server nodes was found to vary from 3 Mbps to 186 Mbps. Bitcoin's block size has also varied over the years, from 87 KB in 2012 to around 1 MB today. These numbers are likely to change as nodes continuously invest in better network infrastructure, compute and storage hardware, and as the Bitcoin community introduces higher-level protocol changes . To optimize the p2p topology in an evolving landscape, it is desirable for a neighborhood-selection protocol to be adaptive to changes, while at the same time have behavioral similarities to optimal topology constructions such as the geometric graph. Perigee is such a protocol; we discuss it next.

1.4 Perigee

Perigee is a decentralized algorithm that adaptively *learns* to form optimal peer connections, purely based on a node's interactions with its neighbors. Unlike hand-crafted protocols which often require extensive manual tuning to optimize protocol parameters for individual blockchain networks, Perigee is a flexible learning algorithm that automatically finds the best topology for any network setting. In Perigee , nodes continuously monitor the promptness of block delivery from each of their neighboring nodes, and decide whether to retain their neighbors or explore connecting to other potentially better-connected peers. Since each node tries to locally find the best set of neighbors it can connect to, Perigee naturally benefits the self-interests of peers. In §1.5 we show through extensive experimental evaluations, that our protocol also globally optimizes block propagation delays under diverse settings.

1.4.1 Algorithm Overview

Perigee operates on top of existing block distribution protocols, and does not change the format of blocks or the gossip protocols used for broadcasting them. Instead it simply decides what is the best set of neighbors for a node to connect to, for a given block size and gossip protocol. Starting from an arbitrary initial set of neighbors (e.g, obtained randomly from a bootstrapping server), a node in Perigee pe-

Algorithm 1 PERIGEE : Algorithm template for updating neighbors of node v after each round.

input: neighbors Γ_v, outgoing neighbors Γ^o_v, set of blocks B mined during the round, and observation set $\mathcal{O}_v$
output: updated set of outgoing neighbors Γ^o_v for next round
// Score each neighbor based on measurements collected in $\mathbb{O}_v$ using a scoring algorithm (see §1.4.2, §1.4.3 for different scoring methods)
score(u) $\leftarrow$ SCORINGALGORITHM($\mathbb{O}_v$), for each ngbr. $u \in \Gamma^o_v$
// Retain subset of d_v neighbors with best score
$\Gamma^o_v \leftarrow \{u \in \Gamma^o_v : \text{score}(u) \in$ best d_v scores of nodes in $\Gamma^o_v\}$
// Additionally connect to e_v random peers for exploration
$\Gamma^o_v \leftarrow \Gamma^o_v \cup (e_v$ randomly chosen neighbors from $V)$

riodically evaluates its current set of neighbors, to decide which neighbors offer the fastest connectivity to the rest of the network. Connections to those neighbors providing a good connectivity are retained, while the rest are disconnected. Additionally, Perigee also periodically connects to a small number of random nodes, as a means to discover previously unknown but potentially well-connected peers.

In Perigee , a neighbor is evaluated purely based on timestamp measurements of when blocks, or advertisements for blocks, were received from the neighbor.[3] Based on these measurements, a real-valued *score* is computed and assigned to each neighbor, which is then used to decide which subset of neighbors to retain. Using block reception times to score neighbors, is a key novelty in Perigee and has several advantages compared to algorithms discussed in §1.3. A node is identified only by its IP address, and not based on auxiliary information such as its geographical location. This makes our algorithm robust to geo-spoofing attacks. Moreover, by explicitly using block arrival times for scoring neighbors, Perigee automatically takes into account heterogeneities, such as variations in link latencies across geographically separated nodes, and variations in hash power. The resulting topology therefore, has a good connectivity to nodes with high hash power, rather than good connectivity in a simpler graph theoretic sense (e.g., low diameter).

To simplify our exposition, we present Perigee (Algorithm 1) under the network model of §5.2. The algorithm proceeds in rounds, where each round spans the time taken to mine and broadcast K unique blocks $B = \{b_1, b_2, ..., b_K\}$ over the p2p network. For a node $v \in V$, let Γ_v denote the set of v's neighbors in G, and let $\Gamma^o_v \subseteq \Gamma_v$ denote v's outgoing neighbors. When a block $b \in B$ is broadcast during a round, we let $t^b_{u,v}$ be the local time at v when b was received from neighbor $u \in \Gamma_v$. We set $t^b_{u,v} = \infty$ if block b was never relayed to v by u. During a round, each node v collects information about when each block was received from its neighbors in the form of an observation set $\mathbb{O}_v = \{(b, u, t^b_{u,v}) : b \in B, u \in \Gamma_v\}$. Note that it is possible for v to hear about a block for the first time from a non-outgoing neighbor. The tuples collected in $\mathbb{O}_v$ allows Perigee to rate how quickly a neighbor relays blocks relative to other neighbors, and retain connections to the best subset of d_v (e.g., $d_v = 6$) neighbors at the end of each round. Neighbors are evaluated using a scoring function, which estimates the maximum delay taken by a neighbor to forward 90% of blocks to the node v. In addition, Perigee also connects to a small number e_v (e.g., $e_v = 2$) of random peers during each round, for discovering previously unknown peers with good connectivity.

We propose two different scoring methods, depending on whether each neighbor is scored individually (§1.4.2), or groups of neighbors are jointly assigned a score (§1.4.3). In the latter case, the score is an estimate of the maximum delay taken by the group of neighbors as a whole to forward 90% of blocks to v. The joint scoring of different neighbor groups is better suited to our objective, since each peer ultimately just desires to receive blocks as fast as possible, regardless of which specific neighbor forwards the block. Certain blocks may be relayed fast by some neighbors, while other blocks are relayed fast by the remaining neighbors—as long as, together, the set of neighboring nodes result in a quick delivery of a majority of blocks, it is beneficial for the node. However, accurately evaluating scores for all possible groups of neighbors is computationally expensive, and therefore we propose faster approximate methods. In our evaluations (§1.5), we find both the independent scoring and approximate joint scoring methods to be competitive.

[3] Our protocol is general, and can also be used with timestamp measurements of transactions received from neighbors.

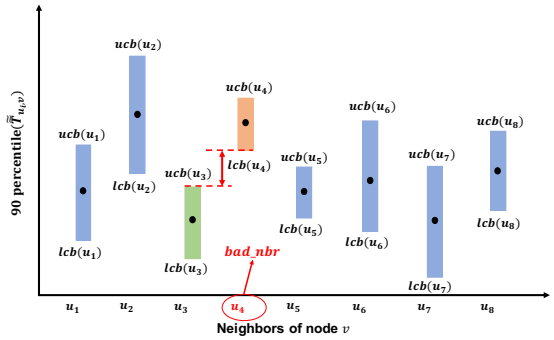


Figure 1-2: Observe that $\text{lcb}(u_4) \geq \text{lcb}(u)\ \forall u \in \Gamma^o_v$, $\text{ucb}(u_3) \leq \text{ucb}(u)\ \forall u \in \Gamma^o_v$. Moreover, $\text{lcb}(u_4) > \text{ucb}(u_3)$ and so the neighbor to be removed (bad_nbr) is u_4.

1.4.2 Scoring Each Neighbor Individually

Vanilla Scoring

We first consider a simple scoring method, called VANILLASCORING, to illustrate how for each neighbor $u \in \Gamma^o_v$ of a node v, the timestamps $t^b_{u,v}$ of blocks $b \in B$ broadcast during a round can be used to estimate u's score. Recall that a timestamp $t^b_{u,v}$ recorded in the observation set $\mathbb{O}_v$ of a node v corresponds to the local wall-clock time when the block b was received at v. In order to judge how well a neighbor is connected to the rest of the network, it is desirable to know the *relative* time between when a block was mined, and when it was delivered by the neighbor (*i.e.,*the propagation delay). However, as it is difficult for a node v to know the precise time when a block was mined, we use the relative time differences between when a block is forwarded by different neighbors, as a proxy for the propagation delay. For a block b, the first time it was received by v from some neighbor is at time $t^b_v := \min_{u \in \Gamma_v} t^b_{u,v}$. The timestamps in $\mathbb{O}_v$ are then revised relative to times blocks were first received by v, and a time-normalized observation set $\tilde{\mathbb{O}}_v$ is computed as

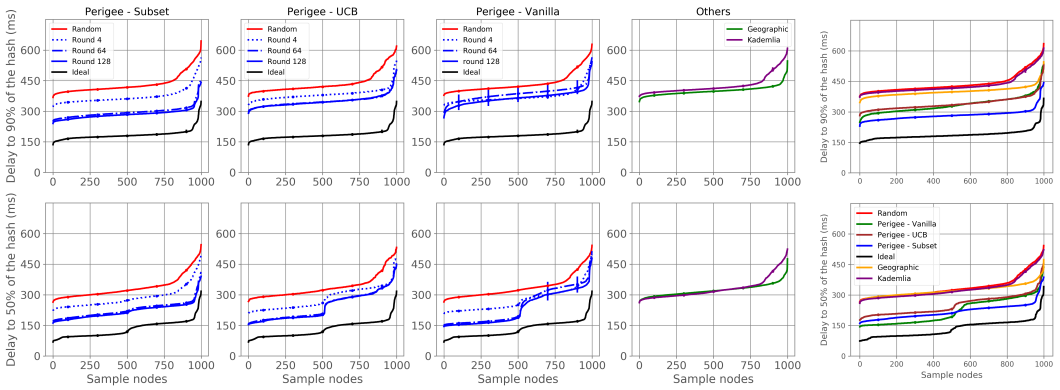
$$\tilde{\mathbb{O}}_v = \{(b, u, t^b_{u,v} - t^b_v) : u \in \Gamma_v, b \in B\}. \tag{1.2}$$

In VANILLASCORING, the score for a neighbor $u \in \Gamma^o_v$ is simply computed as the 90th percentile of the multi-set of relative timestamps $\tilde{T}_{u,v} := (\tilde{t} : (b, u, \tilde{t}) \in \tilde{\mathbb{O}}_v)$ observed in a round. This scoring approach naturally reflects a node's preference to retain an outgoing neighbor from which it received transactions relatively earlier. The lower the score for a neighbor, the higher is the preference for node v to retain the neighbor in next round.

UCB Scoring

In the VANILLASCORING method, propagation delay estimates for individual neighbors (90th percentile of relative timestamp observations) are likely to be noisy if the number of blocks $|B|$ in a round is small. The noise here arises due to the randomness in which node mines a block each time (§1.1.1). While increasing $|B|$ by increasing the duration of each round improves accuracy of our estimates, it also slows down the overall convergence time of the algorithm.[4] To improve the accuracy of the VANILLASCORING estimates, without sacrificing on convergence time, we propose a second scoring method motivated by the Upper Confidence Bound (UCB) algorithm for multi-armed bandits [24]. In the UCBSCORING method, a node maintains an estimate of propagation delay for a neighbor, based on observed timestamps, and also computes lower and upper confidence bounds for it. If a neighbor has been connected to v for longer than one round, then the estimates and confidence bounds for the neighbor are computed not only using the observations $\mathbb{O}_v$ made during the current round, but also using past observations available for the neighbor. For a neighbor $u \in \Gamma^o_v$, let $\tilde{T}_{u,v}(-i)$ denote the multi-set of relative timestamps obtained during a round i rounds before the present round. Supposing node u has been v's neighbor for the past $r_{u,v}$ rounds. In the UCBSCORING approach, we use a multi-set of relative timestamp

[4] We illustrate convergence of Perigee empirically in our experiments in §1.5.



(a) (b)

Figure 1-3: Minimum delay to nodes totaling 90% of network's hash power on random, geographic, Perigee -Vanilla, Perigee -Subset, Perigee -UCB, Kademlia and the fullly-connected graph (denoted as ideal). (a) All nodes have the same hash power. (b) Nodes have a hash power drawn from an exponential distribution.

observations $\tilde{\tilde{T}}_{u,v} = (\tilde{t} : (b,u,\tilde{t}) \in \cup_{i=0}^{-r_{u,v}} \tilde{T}_{u,v}(-i)$ such that $\tilde{t} < \infty)$ for a neighbor u.[5] The propagation delay for u is estimated as the 90th percentile of $\tilde{\tilde{T}}_{u,v}$, and its confidence bounds are computed as

$$\text{ucb}(u) = \texttt{90percentile}(\tilde{\tilde{T}}_{u,v}) + c\sqrt{\frac{\log(|\tilde{\tilde{T}}_{u,v}|)}{2\times|\tilde{\tilde{T}}_{u,v}|}} \tag{1.3}$$

$$\text{lcb}(u) = \texttt{90percentile}(\tilde{\tilde{T}}_{u,v}) - c\sqrt{\frac{\log(|\tilde{\tilde{T}}_{u,v}|)}{2\times|\tilde{\tilde{T}}_{u,v}|}}, \tag{1.4}$$

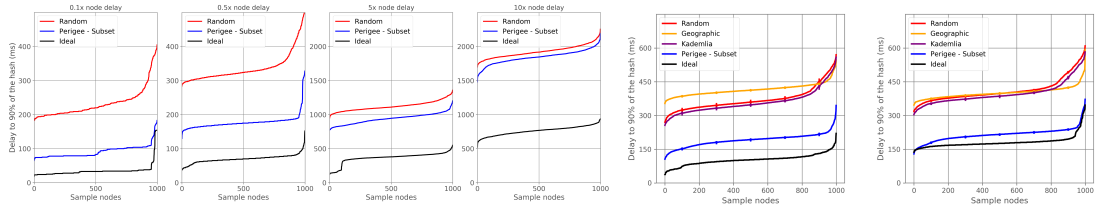
where ucb and lcb denote the upper and lower confidence bounds respectively [24], and $\texttt{90percentile}(\cdot)$ computes the 90th percentile of its argument. At the end of each round, in the UCBScoring approach we check whether $\max_{u\in\Gamma_v^o}(\text{lcb}(u)) > \min_{u\in\Gamma_v^o}(\text{ucb}(u))$, and if so, v disconnects from the neighbor $\text{argmax}_{u\in\Gamma_v^o}(\text{lcb}(u))$ and connects to a random new neighbor instead; otherwise the current set of neighbors are retained for the next round. Figure 1-2 shows an example of upper and lower confidence bounds for a set of eight neighbors. Node u_4 will be disconnected at the end of the round in this example. Updating the set of neighbors this way based on confidence intervals, avoids accidentally disconnecting from a well-connected neighbor that has a poor 90th percentile score due to randomness in mining and lack of sufficient measurement samples.

1.4.3 Scoring Groups of Neighbors Jointly

Next, we present an alternative scoring method, SubsetScoring, where scores are assigned to each group $\gamma_v \subset \Gamma_v^o$ of neighbors (of a certain cardinality, e.g., 6) instead of to individual nodes. At the end of a round, the group of neighbors having the best score are retained and neighbors that are not part of this group are disconnected. As before, a small number of neighboring connections are made randomly in each round to encourage exploration.

To avoid the computational overhead of exhaustively evaluating scores for all possible subsets of neighbors, we consider a simpler, but approximate, greedy approach in which the neighbors to be retained are selected one by one. First, the algorithm selects the neighbor $u_1 \in \Gamma_v^o$ having the best 90th percentile score in the relative timestamp observation multi-set $\tilde{T}_{u,v}$ (§1.4.2). If k neighbors

[5] Note that the union $\cup_{i=0}^{-r_{u,v}} \tilde{T}_{u,v}(-i)$ is a multi-set union.



(a) (b) (c)

Figure 1-4: (a) Delay distributions for Perigee with 0.1×, 0.5×, 5× and 10× the default node delay. (b) Setting with a small number (10%) of high hash power miners. (c) Performance in the presence of a low-latency block distribution network such as bloXroute.

$u_1, u_2, \ldots, u_k$ have been selected, the $(k+1)$st neighbor is selected by first computing a transformed observation set $\widetilde{\widetilde{O}}_v(u_1, u_2, \ldots, u_k) = \{(b, u, \min(\tilde{t}^b_{u,v}, \min_{1 \le i \le k} \tilde{t}^b_{u_i,v})) : b \in B, u \in \Gamma^o_v \backslash \{u_1, u_2, \ldots, u_k\}\}$, followed by the multi-set of relative timestamps $\widetilde{\widetilde{T}}_{u,v}(u_1, \ldots, u_k) = (\widetilde{\widetilde{t}} : (b, u, \widetilde{\widetilde{t}}) \in \widetilde{\widetilde{O}}_v(u_1, \ldots, u_k))$ for each neighbor $u \in \Gamma^o_v \backslash \{u_1, \ldots, u_k\}$. The transformation essentially avoids penalizing nodes that do not have good connectivity to a certain part of the network, to which the neighbors already chosen have a good connectivity. The node $u \in \Gamma^o_v \backslash \{u_1, \ldots, u_k\}$ with the lowest `90percentile`$(\widetilde{\widetilde{T}}_{u,v}(u_1, \ldots, u_k))$ value is finally selected as the $(k+1)$st choice. Thus, each time a neighbor is chosen whose connectivity to the rest of the network best complements the other neighbors selected thus far. As in §1.4.2, once $(d_v - e_v)$ neighbors are selected, node v also randomly selects e_v nodes as part of exploration. This set of d_v nodes are v's updated set of neighbors that it will monitor in the next round.

1.5 Evaluation

We evaluate the performance of Perigee , and compare it against the baseline algorithms of §1.3. Our experiments are based on a Python simulator we built following the network model of §5.2.[6] We describe the experimental setting in §1.5.1. Following this, we evaluate Perigee on a variety of different network conditions (§1.5.2–§1.5.4).

1.5.1 Experimental Setup

Network setting. We retrieved a publicly available list of 9408 Bitcoin nodes [3], and use a randomly sampled subset of 1000 nodes from it, for all our experiments. The "default" setting for hash power of nodes, block validation times, link propagation delay, and block size in our experiments are described below. In §1.5.2, §1.5.3, we consider a broader range of settings for each of hash power, block validation times respectively; in each case, while we explore different settings for one attribute, we fix the other attributes to their "default" setting unless specified otherwise. The default settings are as follows.
(1) *Hash power.* We assume hash power is distributed uniformly across all the nodes.
(2) *Propagation delay.* The dataset of Bitcoin nodes [3] includes information about the geographical location of each node. Nodes are spread across seven geographic regions: North America, South America, Europe, Asia, Africa, China, and Oceania. We set the propagation latency between any two nodes according to their geographical locations, using the iPlane latency measurement dataset [118, 148].
(3) *Block size.* We assume block sizes are small, relative to the bandwidth available at the nodes. Hence the overall block broadcasting delay is dominated by the link propagation delays, and block validation delays, in the default setting.
(4) *Block validation time.* Each node has a mean block processing time of 50 ms.
In addition, in §1.5.4 we consider a scenario where nodes have access to a high-speed block distribution network such as BloXroute [109]. Each node creates 8 outgoing connections, and accepts up to

[6] Source code and datasets are available at `https://github.com/mori94/perigee`.

20 incoming connections. If a node already has 20 incoming connections, any additional connection request is declined by the node.

Algorithms compared. We implement Perigee under the scoring methods discussed in §1.4.2, §1.4.2, §1.4.3, and name them Perigee -Vanilla, Perigee -UCB and Perigee -Subset respectively. For Perigee -Vanilla and Perigee -Subset, we define a round such that $|B| = 100$ blocks are mined during each round; for Perigee -UCB we use shorter rounds in which only one block ($|B|=1$) is mined each round. In all of the Perigee variants, a node selects two neighbors randomly for exploration every round (§1.4). As baseline algorithms, we consider the random connection algorithm (§1.3.1), geography-based connection algorithm (§1.3.2) and a structured p2p topology based on Kademlia [148]. For these baselines, we do not change the topology with each round. We also consider a topology in which each node is connected to all other nodes, to obtain a theoretical lower bound on block propagation times.

Performance metric. For each node, we compute the time it takes for a block broadcast by the node to reach 50% and 90% of the hash power in the network. We repeat each experiment three times using independently sampled link latencies, and plot the mean propagation times for different nodes in ascending order; we also show error bars at the 100th, 300th, 500th, 700th and 900th node. Note that the nodes corresponding to the same x-coordinate value may not be the same node in the network.

1.5.2 Hash Power

We first consider the setting where all the attributes—hash power, link propagation latencies, block validation times, block size—take their default setting (§1.5.1) and plot the results in Figure 1-3(a). The Perigee -Subset and -UCB algorithm achieve around 33% and 11% lower delay respectively compared to random neighbor selection, indicating that switching neighbors based on their scores helps reduce the block propagation delays. Connecting based on node geography does help lower delay compared to random selection, however it is still 40% worse than Perigee -Subset at the 500th node. The Kademlia topology is slightly worse than even the geographic topology. While the 90-percentile delays in Perigee converge as the number of rounds increases, we observe the 50-percentile delays do not exhibit a similar monotonicity. This is because Perigee chooses neighbors only to optimize nodes' 90-percentile delays.

Since Perigee -Subset is slightly better than Perigee -UCB or -Vanilla, for the reminder we have used Perigee -Subset as the preferred scoring method. Next, we consider the same setting as above but where the hash power of the nodes are sampled from an exponential distribution (of mean 1), and normalized to 1 (Figure 1-3(b)). The results show a similar performance pattern as in Figure 1-3(a) with Perigee -Subset being 33% better than random.

1.5.3 Processing Delay

In Figure 1-4, we vary the block validation time to 0.1×, 0.5×, 5× and 10× its default value. As shown in the Figure, for small values of node delay (0.1×), Perigee finds a topology with delays at least 62% better than random. However, as the node delay increases, Perigee approaches the random protocol's performance. This is expected, since with large processing delays the 90th percentile delay is dictated by the number of nodes on the shortest paths to nodes (*i.e.,*the diameter of the network). With node degree bounded by a constant, the diameter is lower-bounded by the logarithm in number of nodes, which is achieved by the random topology.

1.5.4 Fast Distribution Networks

The Bitcoin network is known to have a small number of mining pools that contribute to most of the hash power in the network. To simulate such a network, we randomly select 10% of the nodes and assign them 90% of the network's total hash power; we also set the link propagation latencies between the high-power miners to be much smaller than their default values. In this network, it is desirable for peers to be directly connected to at least one of the high-power miners. As shown by the results in Figure 1-4(b), Perigee can exploit and explore the network to get much closer to the ideal delay in a fully-connected network compared to baselines. We also simulate fast block relay networks, by considering 100 nodes organized as a tree topology with low-propagation-latency links. The block

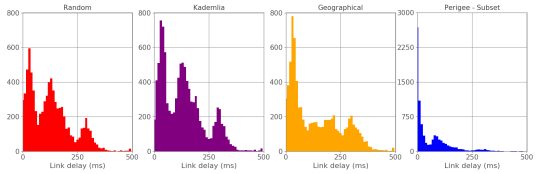


Figure 1-5: Histograms of the edge latencies in the p2p graph obtained after the execution of various algorithms under uniform hash power.

validation delays for these 100 nodes are also set to be 10% of their default value. Even here, as before, our results in Figure 1-4(c) show that Perigee can approach the fully-connected network baseline closely.

1.5.5 What does Perigee learn?

In Figure 1-5 we observe that the distribution of edge latencies of the p2p network obtained in all the four algorithms are bimodal. The lower mode is mostly populated by intra-continental edges with smaller edge latencies whereas the upper mode is mostly populated by inter-continental edges with larger edge latencies. For Perigee -subset, the latencies of bulk of the edges are populated around the lower mode. On the other hand, this is not the case in random and geometric. This implies that over the course of execution of Perigee -subset algorithm, nodes learn to select those outgoing neighbors with which they have smaller edge latency.

Chapter 2

PoSAT: Proof-of-Work Availability and Unpredictability, without the Work

An important feature of Proof-of-Work (PoW) blockchains is full dynamic availability, allowing miners to go online and offline while requiring only 50% of the online miners to be honest. Existing Proof-of-stake (PoS), Proof-of-Space and related protocols are able to achieve this property only partially, either requiring the additional assumption that adversary nodes are online from the beginning and no new adversary nodes come online afterwards, or use additional trust assumptions for newly joining nodes. We propose a new PoS protocol PoSAT which can provably achieve dynamic availability fully without any additional assumptions. The protocol is based on the longest chain and uses a Verifiable Delay Function for the block proposal lottery to provide an arrow of time. The security analysis of the protocol draws on the recently proposed technique of Nakamoto blocks as well as the theory of branching random walks. An additional feature of PoSAT is the complete unpredictability of who will get to propose a block next, even by the winner itself. This unpredictability is at the same level of PoW protocols, and is stronger than that of existing PoS protocols using Verifiable Random Functions.

2.1 Introduction

2.1.1 Dynamic Availability

Nakamoto's invention of Bitcoin [127] in 2008 brought in the novel concept of a permissionless Proof-of-Work (PoW) consensus protocol. Following the longest chain protocol, a block can be proposed and appended to the tip of the blockchain if the miner is successful in solving the hash puzzle. The Bitcoin protocol has several interesting features as a consensus protocol. An important one is *dynamic availability*. Bitcoin can handle an uncertain and dynamic varying level of consensus participation in terms of mining power. Miners can join and leave as desired without any registration requirement. This is in contrast to most classical Byzantine Fault Tolerant (BFT) consensus protocols, which assumes a fixed and known number of consensus nodes. Indeed, Bitcoin has been continuously available since the beginning, a period over which the hashrate has varied over a range of 14 orders of magnitude. Bitcoin has been proven to be secure as long as the attacker has less than 50% of the online hash power (the static power case is considered in [127, 88, 131] and variable hashing power case is considered in [90, 92]).

Recently proof-of-stake (PoS) protocols have emerged as an energy-efficient alternative to PoW. Instead of solving a difficult hash puzzle, nodes participate in a lottery to win the right to append a block to the blockchain, with the probability of winning proportional to a node's stake in the total pool. This replaces the resource intense mining process of PoW, while ensuring fair chances to contribute and claim rewards.

There are broadly two classes of PoS protocols: those derived from classical BFT protocols and those inspired by Nakamoto's longest chain protocol. Attempts at blockchain design via the BFT approach include Algorand [55, 94], Tendermint [39] and Hotstuff [164]. Motivated and inspired by Nakamoto longest chain protocol are the PoS designs of Snow White [32] and the Ouroboros family of protocols [107, 65, 26]. One feature that distinguish the PoS longest chain protocols from the BFT protocols is that they inherit the dynamic availability of Bitcoin: the chain always grows regardless of the number of nodes online. But do these PoS longest chain protocols provide the same level of security guarantee as PoW Bitcoin in the dynamic setting?

2.1.2 Static vs Dynamic Adversary

Two particular papers focus on the problem of dynamic availability in PoS protocols: the sleepy model of consensus [136] and Ouroboros Genesis [26]. In both papers, it was proved that their protocols are secure if less than 50% of the online nodes are adversary. This condition is the same as the security guarantee in PoW Bitcoin, but there is an additional assumption: *all adversary nodes are always online starting from genesis and no new adversary nodes can join.* While this static adversary assumption seems reasonable (why would an adversary go to sleep?), in reality this can be a very restrictive condition. In the context of Bitcoin, this assumption would be analogous to the statement that the hash power of the adversary is fixed in the past decade (while the total hashing power increased 14 orders of magnitude!) More generally, in public blockchains, PoW or PoS, no node is likely to be adversarial during the launch of a new blockchain token - adversaries only begin to emerge later during the lifecycle.

The static adversary assumption underlying these PoS protocols is not superfluous but is in fact *necessary* for their security. Suppose for the 1^{st} year of the existence of the PoS-based blockchain, only 10% of the total stake is online. Out of this, consider that all nodes are honest. Now, at the beginning of the 2^{nd} year, all 100% of the stake is online out of which 20% is held by adversary. At any point of time, the fraction of online stake held by honest nodes is greater than 0.8. However, both Sleepy and Genesis are not secure since the adversary can use its 20% stake to immediately participate in all past lotteries to win blocks all the way back to the genesis and then grow a chain *instantaneously* from the genesis to surpass the current longest chain (Figure 2-1(a)). Thus, due to this "costless simulation", newly joined adversary nodes not only increase the current online adversary stake, but effectively increase past online adversary stake as well. See Appendix 2.6.3 for further details on how costless simulation renders both sleepy model of consensus and Ouroboros Genesis vulnerable to attacks. In contrast, PoW does not suffer from the same issue because it would take a long time to grow such a chain from the past and that chain will always be behind the current longest chain. Thus, PoW provides an *arrow of time*, meaning nodes cannot "go back in time" to mine blocks for the times at which they were not online. This property is key in endowing PoW protocols with the ability to tolerate fully dynamic adversaries wherein both honest nodes and adversary can have varying participation (Figure 2-1(b)).

We point out that some protocols including Ouroboros Praos [65] and Snowhite [32] require that nodes discard chains that fork off too much from the present chain. This feature was introduced to handle nodes with expired stake (or nodes that can perform key grinding) taking over the longest chain. While they did not specifically consider the dynamic adversary issue we highlighted, relying on previous checkpoints can potentially solve the aforementioned security threat. However, as was eloquently argued in Ouroboros Genesis [26], these checkpoints are unavailable to offline clients and newly joining nodes require advice from a trusted party (or a group inside which a majority is trusted). This trust assumption is too onerous to satisfy in practice and is not required in PoW. Ouroboros Genesis was designed to require no trusted joining assumption while being secure to long-range and key-grinding attacks. However, they are not secure against dynamic participation by the adversary: they are vulnerable to the aforementioned attack. This opens the following question:

Is there a fully dynamically available PoS protocol which has full PoW security guarantee, without additional trust assumptions?

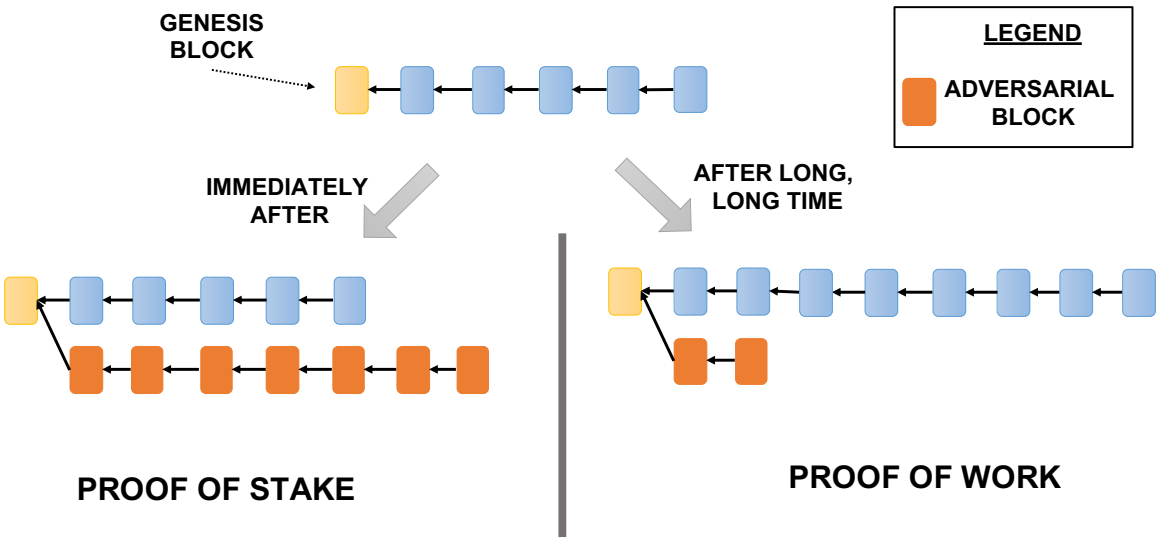


Figure 2-1: (a) Newly joined nodes in existing PoS protocols can grow a chain from genesis instantaneously. (b) Newly joined miners in PoW protocol takes a long time to grow such a chain and is always behind.

2.1.3 PoSAT achieves PoW dynamic availability

We answer the aforementioned question in the affirmative. Given that arrow-of-time is a central property of PoW protocols, we design a new PoS protocol, PoS with Arrow-of-Time (PoSAT), also having this property using randomness generated from Verifiable Delay Functions (VDF). VDFs are built on top of iteratively sequential functions, i.e., functions that are only computable sequentially: $f^{\ell}(x) = f \circ f \circ ... \circ f(x)$, along with the ability to provide a short and easily verifiable proof that the computed output is correct. Examples of such functions include (repeated) squaring in a finite group of unknown order [46, 147], i.e,, $f(x) = 2^x$ and (repeated) application of secure hash function (SHA-256) [119], i.e,, $f(x) = \text{HASH}(x)$. While VDFs have been designed as a way for proving the passage of a certain amount of time (assuming a bounded CPU speed), it has been recently shown that these functions can also be used to generate an unpredictable randomness beacon [73]. Thus, running the iteration till the random time L when $\text{RANDVDF}(x) = f^L(x) < \tau$ is within a certain threshold will result in L being a geometric random variable. We will incorporate this randomized VDF functionality to create an arrow-of-time in our protocol.

The basic idea of our protocol is to mimic the PoW lottery closely: instead of using the solution of a Hash puzzle based on the parent block's hash as proof of work, we instead use the randomized VDF computed based on the parent block randomness and the coin's public key as the proof of stake lottery. In a PoW system, we are required to find a string called "`nonce`" such that $\text{HASH}(\texttt{block},\texttt{nonce}) < \tau$, a hash-threshold. Instead in our PoS system, we require $\text{RANDVDF}(\texttt{randSource},pk,\texttt{slot}) < \tau$, where `randSource` is the randomness from the parent block, pk is the public key associated with the mining coin and `slot` represents the number of iterations of the RANDVDF since genesis. There are four differences, the first three are common in existing PoS systems: (1) we use "`randSource`" instead of "`block`" in order to prevent grinding attacks on the content in the PoS system, (2) we use the public-key "pk" of staking coin instead of PoW "`nonce`" to simulate a PoS lottery, (3) we use "`slot`" for ensuring time-ordering, (4) instead of using a HASH, we use the RANDVDF, which requires sequential function evaluation thus creating an "arrow of time".

The first two aspects are common to many PoS protocols and is most similar to an earlier PoS protocol [78], however, crucially we use the RANDVDF function instead of a Verifiable random function (VRF) and a time parameter inside the argument used in that protocol. This change allows for full dynamic availability: if adversaries join late, they cannot produce a costless simulation of the time that they were not online and build a chain from genesis instantaneously. It will take the adversary

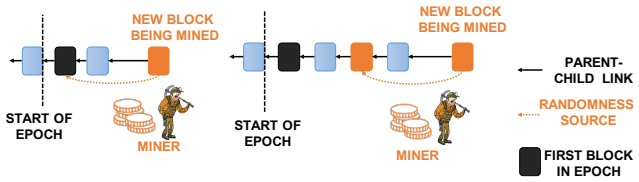


Figure 2-2: Left: A node uses randomness from the first block of the epoch. Right: Since a node already won a block in the period, it uses that block's randomness.

time to grow this chain (due to the sequential nature of the RANDVDF), by which time, the honest chain would have grown and the adversary will be unable to catch up. Thus, PoSAT behaves more like PoW (Figure 2-1(b)) rather than existing PoS based on VRF's (Figure 2-1(a)). We show that this protocol achieves full dynamic availability: if $\lambda_h(t)$ denotes the honest stake online at t, $\lambda_a(t)$ denotes the online adversarial stake at time t, it is secure as long as

$$\lambda_h(t) > e\lambda_a(t) \qquad \text{for all } t, \tag{2.1}$$

where e is Euler's number 2.7182....

We observe that the security of this protocol requires a stronger condition than PoW protocols. The reason for this is that an adversary can potentially do parallel evaluation of VDF on *all* possible blocks. Since the randomness in each of the blocks is independent from each other, the adversary has many random chances to increase the chain growth rate to out-compete the honest tree. This is a consequence of the nothing-at-stake phenomenon: the same stake can be used to grind on the many blocks. The factor e is the resulting amplification factor for the adversary growth rate. This is avoided in PoW protocols due to the conservation of work inherent in PoW which requires the adversary to split its total computational power among such blocks.

We solve this problem in PoSAT by reducing the rate at which the block randomness is updated and hence reducing the block randomness grinding opportunities of the adversary. Instead of updating the block randomness at every level of the blocktree, we only update it once every c levels (called an epoch). The larger the value of the parameter c, the slower the block randomness is updated. The common source of randomness used to run the VDF lottery remains the same for c blocks starting from the genesis and is updated only when (a) the current block to be generated is at a depth that is a multiple of c, or (b) the coin used for the lottery is successful within the epoch of size c. The latter condition is necessary to create further independent winning opportunities for the node within the period c once a slot is obtained with that coin. This is illustrated in Figure 2-2. For $c=1$, this corresponds to the protocol discussed earlier.

The following security theorem is proved about PoSAT for general c, giving a condition for security (liveness and persistence) under *all* possible attacks.

Theorem 3 (Informal)**.** *PoSAT with parameter c is secure as long as*

$$\frac{\lambda_h^c(t)}{1+\lambda_{\max}\Delta} > \phi_c\lambda_a(t) \qquad \textit{for all } t, \tag{2.2}$$

where $\lambda_h^c(t)$ is the honest stake this is online at time t and has been online since at least $t-\Theta(c)$, Δ is the network delay between honest nodes, $\lambda_{\max}$ is a constant such that $\lambda_h^c(t) \le \lambda_{\max}$ for all $t>0$, ϕ_c is a constant, dependent on c, given in (2.22). $\phi_1 = e$ and $\phi_c \to 1$ as $c\to\infty$.

c	1	2	3	4	5	6	7	8	9	10
ϕ_c	e	2.22547	2.01030	1.88255	1.79545	1.73110	1.68103	1.64060	1.60705	1.57860
$\frac{1}{1+\phi_c}$	$\frac{1}{1+e}$	0.31003	0.33219	0.34691	0.35772	0.36615	0.37299	0.37870	0.38358	0.38780

Table 2.1: Numerically computed values of the adversary amplification factor ϕ_c. The ratio $1/(1+\phi_c)$ is the adversarial fraction of stake that can be tolerated by PoSAT when $\Delta=0$.

We remark that in our PoS protocol, we have a known upper bound on the rate of mining blocks (by assuming that the entire stake is online). We can use this information to set $1+\lambda_{\max}\Delta$ as close to 1 as desired by simply setting the mining threshold appropriately. Furthermore, by setting c large, $\phi_c \approx 1$ and thus PoSAT can achieve the same security threshold as PoW under full dynamic availability. The constant ϕ_c is the amplification of the adversarial chain growth rate due to nothing-at-stake, which we calculate using the theory of branching random walks[153]. The right hand side of (2.2) can therefore be interpreted as the growth rate of a private adversary tree with the adversary mining on every block. Hence, condition (2.2) can be interpreted as the condition that the private Nakamoto attack [127] does not succeed. However, Theorem 4 is a *security theorem*, i.e. it gives a condition under which the protocol is secure under *all* possible attacks. Hence what Theorem 4 says is therefore that among all possible attacks on PoSAT , the private attack is the *worst* attack. We prove this by using the technique of blocktree partitioning and Nakamoto blocks, introduced in [69], which reduce all attacks to a union of private attacks.

We note that large c is beneficial from the point of view of getting a tight security threshold. However, we do require c to be finite (unlike other protocols like Ouroboros that continue to work under c being infinite). This is because the latency to confirm a transaction increases linearly in c (see Section 2.4). Furthermore, an honest node on coming online has to wait until encountering the next epoch beginning before it can participate in proposing blocks and the worst-case waiting time increases linearly with c. We note that the adversary cannot use the stored blocks in the next epoch, thus having a bounded reserve of blocks. The total number of blocks stored up by an adversary potentially increases linearly in the epoch size, thus requiring the confirmation depth and thus latency to be larger than $\Theta(c)$. By carefully bounding this enhanced power of the adversary, for any finite c, we show that PoSAT is secure.

Assuming $\lambda_{\max}\Delta$ to be small, the comparison of PoSAT with other protocols is shown in Table 2.1.3. Here we use Λ_a to be the largest adversary fraction of the total stake online at any time during the execution ($\Lambda_a = \sup_t \lambda_a(t)$). Protocols whose security guarantee assumes all adversary nodes are online all the time effectively assumes that $\lambda_h(t) > \Lambda_a$. Thus existing protocols have limited dynamic availability.

	Ourboros	Snow White / Praos	Genesis / Sleepy	Algorand	PoSAT
Dynamic Availability	$\lambda_h(t) > \Lambda_a$	$\lambda_h(t) > \Lambda_a$	$\lambda_h(t) > \Lambda_a$	No	$\lambda_h^c(t) > \phi_c \lambda_a(t)$
Predictability	Global	Local	Local	Local	None

2.1.4 PoSAT has PoW Unpredictability

Another key property of PoW protocols is their ability to be unpredictable: no node (including itself) can know when a given node will be allowed to propose a block ahead of the proposal slot. We point out that PoSAT with any parameter c remains unpredictable due to the the unpredictability of the RandVDF till the threshold is actually reached. We refer the reader to Fig. 2-2(a) where if the randomness source is at the beginning of the epoch it is clear that the unpredictability of the randomized VDF implies unpredictability in our protocol. However, in case the miner has already created a block within the epoch (Fig. 2-2(b)), the randomness source is now her previous block. This can be thought of as a continuation of the iterative sequential function from the beginning of the epoch and hence it is also unpredictable as to when the function value will fall below a threshold. Thus PoSAT achieves true unpredictability, matching the PoW gold standard, where even an all-knowing adversary has no additional predictive power.

The first wave of PoS protocols such as Ouroboros [107] are fully predictable as they rely on mechanisms for proposer election that provide global knowledge of all proposers in an epoch ahead of time. The concept of Verifiable Random Functions (VRF), developed in [124, 70], was pioneered in the blockchain context in Algorand [55, 94], as well as applied in Ouroboros Praos [65] and Snow White [32]. The use of a private leader election using VRF enables no one else other than the proposer to know of the slots when it is allowed to propose blocks. However, unlike Bitcoin, the proposer itself can predict. Thus, these protocols still allow *local* predictability. The following vulnerability is caused by local predictability: a rational node may then willingly sell out his slot to an adversary. In Ouroboros

Praos, such an all-knowing adversary needs to corrupt only 1 user at a time (the proposer) adaptively in order to do a double-spend attack. He will first let the chain build for some time to confirm a transaction, and then get the bribed proposers one at a time to build a competing chain. Algorand is more resilient, but even there, in each step of the BFT algorithm, a different committee of nodes is selected using a VRF based sortition algorithm. These nodes are locally predictable as soon as the previous block is confirmed by the BFT - and thus an all-knowing adversary only needs to corrupt a third of a committee. Assuming each committee is comprised of K nodes (K being a constant), the adversary only needs to corrupt $\frac{K}{3N}$ fraction of the nodes. Refer to Appendix 2.6.4 for further details.

We summarize the predictability of various protocols in Table 2.1.3.

2.1.5 Related Work

Our design is based on frequent updates of randomness to run the VDF lottery. PoS protocols that update randomness at each iteration have been utilized in practice as well as theoretically proposed [78] - they do not use VDF and have **neither** dynamic availability nor unpredictability. Furthermore, they still face nothing-at-stake attacks. In fact, the amplification factor of e we discussed earlier has been first observed in a Nakamoto private attack analysis in [78]. This analysis was subsequently extended to a full security analysis against *all* attacks in [160, 69], where it was shown that the private attack is actually the worst attack. In [160], the idea of c-correlation was introduced to reduce the rate of randomness update and to reduce the severity of the nothing-at-stake attack; we borrowed this idea from them in the design of our VDF-based protocol, PoSAT .

There have been attempts to integrate VDF into the proof-of-space paradigm [56] as well as into the proof-of-stake paradigm [25], [115], all using a VRF concatenated with a VDF. But, in [56], the VDF runs for a fixed duration depending on the input and hence is predictable, and furthermore do not have security proofs for dynamic availability. In [25], the randomness beacon is not secure till the threshold of $1/2$ as claimed by the authors since it has a randomness grinding attack which can potentially expand the adverarial power by at least factor e. There are three shortcomings in [115] as compared to our paper: (1) even under static participation, they only focus on an attack where an adversary grows a private chain, (2) there is no modeling of dynamic availablility and a proof of security and (3) since the protocol focuses only on $c=1$, they can only achieve security till threshold $1/1+e$, not till $1/2$. We note that recent work [38] formalized that a broad class of PoS protocols suffer from either of the two vulnerabilities: (a) use recent randomness, thus being subject to nothing-at-stake attacks or (b) use old randomness, thus being subject to prediction based attacks (even when only locally predictable). We note that PoSAT with large c completely circumvents both vulnerabilities using the additional VDF primitive since it is able to use old randomness while still being fully unpredictable.

We want to point out that dynamic availability is distinct and complementary to *dynamic stake*, which implies that the set of participants and their identities in the mining is changing based on the state of the blockchain. We note that there has been much existing work addressing issues on the dynamic stake setting - for example, the s-longest chain rule in [26], whose adaptation to our setting we leave for future work. We emphasize that the dynamic availability problem is well posed even in the static stake setting (the total set of stakeholders is fixed at genesis).

2.1.6 Outline

The rest of the paper is structured as follows. Section 2.2 presents the VDF primitive we are using and the overall protocol. Section 5.2 presents the model. Section 2.4 presents the details of the security analysis.

2.2 Protocol

2.2.1 Primitives

In this section, we give an overview of VDFs and refer the reader to detailed definitions in Appendix 2.7.

Definition 1 (from [36]). *A* VDF $V=(\textsc{Setup},\textsc{Eval},\textsc{Verify})$ *is a triple of algorithms as follows:*

- SETUP(λ,τ)$\rightarrow$**pp**=(ek,vk) *is a randomized algorithm that produces an evaluation key* ek *and a verification key* vk.
- EVAL(ek,$input$,τ)$\rightarrow$(O,$proof$) *takes an input*$\in\mathcal{X}$, *an evaluation key* ek, *number of steps* τ *and produces an output* $O\in\mathcal{Y}$ *and a (possibly empty)* $proof$.
- VERIFY(vk,$input$,O,$proof$,τ)$\rightarrow Yes$,No *is a deterministic algorithm takes an input, output, proof,* τ *and outputs* Yes *or* No.

VDF.EVAL is usually comprised of sequential evaluation: $f^{\ell}(x) = f \circ f \circ ... \circ f(x)$ along with the ability to provide a short and easily verifiable proof. In particular, there are three separate functions VDF.START, VDF.ITERATE and VDF.PROVE (the first function is used to initialize, the second one operates for the number of steps and the third one furnishes a proof). This is illustrated in Figure 2-3 on the left. While VDFs have been designed as a way for proving the passage of a certain amount of time, it has been recently shown that these functions can also be used to generate an unpredictable randomness beacon [73]. Thus, running the iteration till the random time L when RANDVDF$(x)=f^{L}(x)<\tau$ generates the randomness beacon. This is our core transformation to get a randomized VDF. This is shown in Figure 2-4 on the right. Instead of running for a fixed number of iterations, we run the VDF iterations till it reaches a certain threshold. Our transformation is relatively general purpose and most VDFs can be used with our construction. For example, a VDF (which is based on squaring in a group of unknown order) is an ideal example for our construction [138, 162]. In the recent paper [73], for that sequential function, a new method for obtaining a short proof whose complexity does not depend (significantly) on the number of rounds is introduced - our protocol can utilize that VDF as well. They show furthermore that they obtain a continuous VDF property which implies that partial VDF computation can be continued by a different party - we do not require this additional power in our protocol.

For the RANDVDF in PoSAT , as illustrated in Fig 2-4, `slot` plays a similar role as the timestamps in other PoS protocols like [136]. The `slot` basically mentions the number of times the RANDVDF has iterated since the genesis and when the speed of the iteration of RANDVDF is constant, `slot` is an approximation to the time elapsed since the beginning of the operation of the PoS system.

Normally, a VDF will satisfy *correctness* and *soundness*. And we require RANDVDF to also satisfy correctness and soundness as defined in Appendix 2.7.

A key feature of VDF is that if the VDF takes T steps, then the prover should be able to complete the proof in time (nearly) proportional to T and the verifier should be able to verify the proof in (poly)-logarithmic time. This makes it feasible for any node that receives a block to quickly verify that the VDF in the header is indeed correctly computed, without expending the same effort that was expended by the prover. We refer the reader interested in a detailed analysis of these complexities to Section 6.2 in [138] for the efficiency calculation or Section 2.3 in [73].

2.2.2 Protocol description

The pseudocode for the PoSAT is given in Algorithm 2.

Initialization.

An honest coin n on coming online, calls INITIALIZE() where it obtains the current state of the blockchain, `blkTree`, by synchronizing with the peers via SYNC() and initializes global variables. However, the coin n can start participating in the leader election only after encountering the next epoch beginning, that is, when the depth of the `blkTree` is a multiple of c. This is indicated by setting $\texttt{participate}_n$ to False. Observe that if the coin n is immediately allowed to participate in leader election, then, the coin n would have to initiate RANDVDF.EVAL from the `randSource` contained in the block at the beginning of the current epoch. Due to the sequential computation in RANDVDF, the coin n would never be able to participate in the leader elections for proposing blocks at the tip of the blockchain. In parallel, the coin keeps receiving messages and processes them in RECEIVEMESSAGE(). On receiving a valid block that indicates epoch beginning, $\texttt{randSource}_n$, $\texttt{slot}_n$ and $\texttt{participate}_n$ are updated accordingly (lines 27, 33, 34) for active participation in leader election.

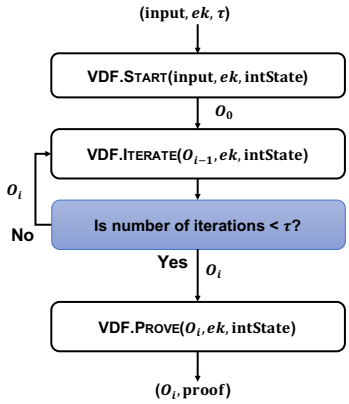


Figure 2-3: VDF.EVAL(input,ek,τ) requires the number of iterations that VDF.ITERATE should run.

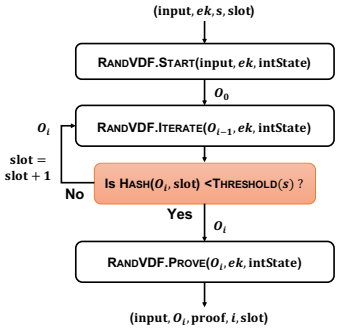


Figure 2-4: On the other hand, RANDVDF.EVAL(input,ek,s,slot) requires the expected number of number of iterations RANDVDF.ITERATE (denoted by s) must run.

Leader election.

The coin n records the tip of the longest chain of blkTree in $\texttt{parentBlk}_n$ (line 25) and contests leader election for appending block to it. RANDVDF.EVAL($\texttt{input}_n$, RANDVDF.ek_n, $\texttt{s}_n$) is used to compute an unpredictable randomness beacon that imparts unpredictability to leader election. The difficulty parameter $\texttt{s}_n$ is set proportional to the current $\texttt{stake}_n$ of the coin n using UPDATETHRESHOLD($\texttt{stake}_n$) and $\texttt{randSource}_n$ is taken as $\texttt{input}_n$. RANDVDF.EVAL($\texttt{input}_n$,ek_n,$\texttt{s}_n$,$\texttt{slot}_n$) is an iterative function composed of:

- RANDVDF.START($\texttt{input}_n$,RANDVDF.ek_n,$\texttt{IntState}_n$) initializes the iteration by setting initial value of $\texttt{output}_n$ to be $\texttt{input}_n$. Note that $\texttt{IntState}_n$ is the internal state of the RANDVDF.

- RANDVDF.ITERATE($\texttt{output}_n$, RANDVDF.ek_n, $\texttt{IntState}_n$) is the iterator function that updates $\texttt{output}_n$ in each iteration. At the end of each iteration, it is checked whether HASH($\texttt{output}_n$,$\texttt{slot}_n$) is less than THRESHOLD($\texttt{s}_n$), which is set proportional to $\texttt{s}_n$. If No, $\texttt{slot}_n$ is incremented by 1 and current $\texttt{output}_n$ is taken as input to the next iteration. If Yes, then it means coin n has won the leader election and $\texttt{output}_n$ is passed as input to RANDVDF.PROVE(.). Observe that the number of iterations, $\texttt{randIter}_n$, that would be required to pass this threshold is unpredictable which lends to randomness beacon. Recall that $\texttt{slot}_n$ is a counter for number of iterations since genesis. In a PoS protocol, it is normally ensured that the timestamps contained in each block of a chain are ordered in ascending order. Here, in PoSAT , instead we ensure that the slot in the blocks of a chain are ordered, irrespective of who proposed it. This is referred to as *time-ordering*. The reader can refer to Appendix 2.6.5 and 2.6.6 for further details on what attacks can transpire if time-ordering is not ensured. The rationale behind setting THRESHOLD($\texttt{s}_n$) proportional to $\texttt{s}_n$

is that even if the stake $\mathtt{s}_n$ is sybil over multiple coins, the probability of winning leader election in at least one coin remains the same. See Appendix 2.6.2 for detailed discussion.

- RandVDF.Prove($\mathtt{output}_n$,RandVDF.ek_n,$\mathtt{IntState}_n$) operates on $\mathtt{output}_n$ using RandVDF.ek_n and $\mathtt{IntState}_n$ to generate $\mathtt{proof}_n$ that certifies the iterative computation done in the previous step.

The source of randomness $\mathtt{randSource}_n$ can be updated in two ways:

- a block, proposed by another coin, at epoch beginning is received (line 27)
- if coin n wins a leader election and proposes its own block (line 15).

While computing RandVDF.Eval(.), if a block is received that updates $\mathtt{parentBlk}_n$, then, RandVDF.Reset() (line 31) pauses the ongoing computation, updates $\mathtt{s}_n$ and continues the computation with updated Threshold($\mathtt{s}_n$). If $\mathtt{randSource}_n$ is also updated, then, RandVDF.Reset() stops the ongoing computation of RandVDF.Eval(.) and calls PoSLeaderElection().

Content of the block.

Once a coin is elected as a leader, all unconfirmed transactions in its buffer are added to the `content`. The `content` also includes the identity $\mathtt{coin}_n$, $\mathtt{input}_n$, $\mathtt{randSource}_n$, $\mathtt{proof}_n$, $\mathtt{randIter}_n$, $\mathtt{slot}_n$ from RandVDF.Eval(.). The `state` variable in the content contains the hash of parent block, which ensures that the content of the parent block cannot be altered. Finally, the header and the content is signed with the secure signature SIGN.sk_n and the block is proposed. When the block is received by other coins, they check that the time-ordering is maintained (line 38) and verify the work done by the coin n using RandVDF.Verify(.) (line 43). Note that the leader election is independent of the content of the block and content of previous blocks. This follows a standard practice in existing PoS protocols such as [26] and [136] for ensuring that a grinding attack based on enumerating the transactions won't be possible. The reader is referred to Appendix 2.6.1 for further details. However, this allows the adversary to create multiple blocks with the same header but different content. Such copies of a block with the same header but different contents are known as a "forkable string" in [107]. We show in the section 2.4 that the PoSAT is secure against all such variations of attacks.

Confirmation rule.

A block is confirmed if the block is $k-$deep from the tip of the longest chain. The value of k is determined by the security parameter.

2.3 Model

We will adopt a continuous-time model. Like the Δ-synchronous model in [131], we assume there is a bounded communication delay Δ seconds between the honest nodes (the particular value of latency of any transmission inside this bound is chosen by the adversary).

The blockchain is run on a network of N honest nodes and a set of adversary nodes. Each node holds a certain number of coins (proportional to their stake). We allow nodes to join and leave the network, thus the amount of honest/adversarial stake which is participating in the protocol varies as a function of time. Recall that, as described in section 2.2, a coin coming online can only participate in the leader election after encountering the next epoch beginning. Let $\lambda_h(t)$ be defined as the stake of the honest coins that are online at time t and has encountered at least one epoch beginning. Thus, $\lambda_h(t)$ is the rate at which honest nodes win leader elections. Let $\lambda_a(t)$ be the stake controlled by the adversary. We will assume there exist constants $\lambda_{\min},\lambda_{\max}>0$ such that

$$\lambda_{\min} \leq \lambda_h(t) \leq \lambda_{\max} \quad \forall \quad t \geq 0. \tag{2.3}$$

The existence of $\lambda_{\max}$ is obvious since we are in a proof-of-stake system, and $\lambda_{\max}$ denotes the rate at which the leader elections are being won if every single stakeholder is online. We need to assume a minimum $\lambda_h(t)$ in order to guarantee that within a bounded time, a new block is created.

An honest node will construct and publicly reveal the block immediately after it has won the corresponding leader election. However, an adversary can choose to not do so. By "private block", we refer to a block whose corresponding computation of RandVDF.Eval was completed by the adversary earlier than when the block was made public. Also, by "honest block proposed at time t", we mean that the computation of RandVDF.Eval was completed at time t and then the associated honest block was instantaneously constructed and publicly revealed.

The evolution of the blockchain can be modeled as a process $\{(\mathcal{T}(t),\mathcal{C}(t),\mathcal{T}^{(p)}(t),\mathcal{C}^{(p)}(t)) : t \geq 0, 1 \leq p \leq N\}$, N being the number of honest nodes, where:

- $\mathcal{T}(t)$ is a tree, and is interpreted as the *mother tree* consisting of all the blocks that are proposed by both the honest and the adversary nodes up until time t (including private blocks at the adversary).
- $\mathcal{T}^{(p)}(t)$ is an induced (public) sub-tree of the mother tree $\mathcal{T}(t)$ in the view of the p-th honest node at time t.
- $\mathcal{C}^{(p)}(t)$ is the longest chain in the tree $\mathcal{T}^{(p)}(t)$, and is interpreted as the longest chain in the local view of the p-th honest node.
- $\mathcal{C}(t)$ is the common prefix of all the local chains $\mathcal{C}^{(p)}(t)$ for $1 \leq p \leq N$.

The process evolution is as follows.

- **M0**: $\mathcal{T}(0) = \mathcal{T}^{(p)}(0) = \mathcal{C}^{(p)}(0), 1 \leq p \leq N$ is a single root block (genesis).
- **M1**: There is an independent leader election at every epoch beginning, i.e., at every block in the blocktree at level $c, 2c, ..., \ell c, ...$ The leader elections are won by the adversary according to independent Poisson processes of rate $\lambda_a(t)$ at time t, one for every block at the aforementioned levels. The adversary can use the leader election won at a block at level ℓc at time t to propose a block at every block in the next $c-1$ levels $\ell c, \ell c+1, ..., \ell c+c-1$ that are present in the tree $\mathcal{T}(t)$. We refer the reader to Figure 2-5 for a visual representation.
- **M2**: Honest blocks are proposed at a total rate of $\lambda_h(t)$ at time t across all the honest nodes at the tip of the chain held by the mining node p, $\mathcal{C}^{(p)}(t)$.
- **M3**: The adversary can replace $\mathcal{T}^{(p)}(t^-)$ by another sub-tree $\mathcal{T}^{(p)}(t)$ from $\mathcal{T}(t)$ as long as the new sub-tree $\mathcal{T}^{(p)}(t)$ is an induced sub-tree of the new tree $\mathcal{T}^{(p)}(t)$, and can update $\mathcal{C}^{(p)}(t^-)$ to a longest chain in $T^{(p)}(t)$. [1]

We highlight the capabilities of the adversary in this model:

- **A1**: Can choose to propose block on multiple blocks of the tree $\mathcal{T}(t)$ at any time.
- **A2**: Can delay the communication of blocks between the honest nodes, but no more than Δ time.
- **A3**: Can broadcast private blocks at times of its own choosing: when private blocks are made public at time t to node p, then these blocks are added to $\mathcal{T}^{(p)}(t^-)$ to obtain $\mathcal{T}^{(p)}(t)$. Note that, under Δ-synchronous model, when private blocks appear in the view of some honest node p, they will also appear in the view of all other honest nodes by time $t+\Delta$.
- **A4**: Can switch the chain where the p-th honest node is proposing block, from one longest chain to another of equal length, even when its view of the tree does not change, i.e., $\mathcal{T}^{(p)}(t) = \mathcal{T}^{(p)}(t^-)$ but $\mathcal{C}^{(p)}(t) \neq \mathcal{C}^{(p)}(t^-)$.

[1] All jump processes are assumed to be right-continuous with left limits, so that $\mathcal{C}(t), \mathcal{T}(t)$ etc. include the new arrival if there is a new arrival at time t.

It is to be noted that we don't consider the adversary to be *adaptive* in the sense that, although adversarial and honest nodes can join or leave the system as they wish, an adversary can never turn honest nodes adversarial. In order to defend against an adaptive adversary, key evolving signature schemes can be used [65]. However, in order to keep the system simple, we don't consider adaptive adversary.

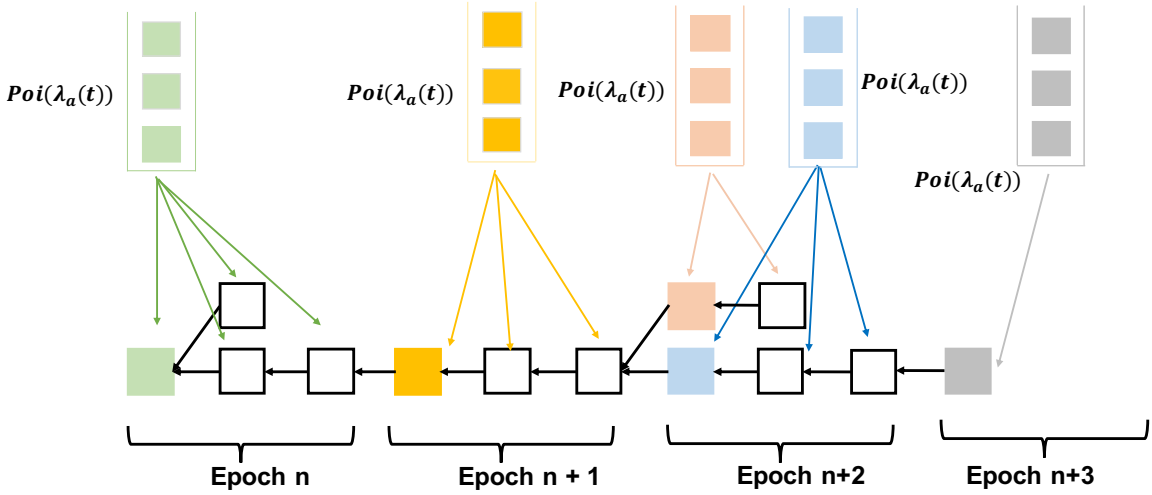


Figure 2-5: There is a separate randomness generated for every block in the modulo c position. Blocks generated from that randomness at time t can attach to any block inside the next $c-1$ blocks that are present in the tree $\mathcal{T}(t)$.

Proving the security (persistence and liveness) of the protocol boils down to providing a guarantee that the chain $\mathcal{C}(t)$ converges fast as $t \to \infty$ and that honest blocks enter regularly into $\mathcal{C}(t)$ regardless of the adversary's strategy.

2.4 Security Analysis

Our goal is to generate a transaction ledger that satisfies persistence and liveness as defined in [88]. Together, persistence and liveness guarantee robust transaction ledger; honest transactions will be adopted to the ledger and be immutable.

Definition 2 (from [88]). *A protocol Π maintains a robust public transaction ledger if it organizes the ledger as a blockchain of transactions and it satisfies the following two properties:*

- *(Persistence) Parameterized by $\tau \in \mathbb{R}$, if at a certain time a transaction* tx *appears in a block which is mined more than τ time away from the mining time of the tip of the main chain of an honest node (such transaction will be called confirmed), then* tx *will be confirmed by all honest nodes in the same position in the ledger.*
- *(Liveness) Parameterized by $u \in \mathbb{R}$, if a transaction* tx *is received by all honest nodes for more than time u, then all honest nodes will contain* tx *in the same place in the ledger forever.*

2.4.1 Main security result

To state our main security result, we need to define some basic notations.

Recall that, as described in section 2.2, a coin coming online can only participate in the leader election after encountering the next epoch beginning. This incurs a random waiting delay for the coin before it can actively participate in the evolution of the blockchain. Hence, the honest mining rate $\lambda_h(t)$, defined in Section 5.2 as the stake of the honest coins that are online at time t and has encountered at least one epoch beginning, is a (random) process that depends on the dynamics of the blockchain.

Hence, we cannot state a security result based on conditions on $\lambda_h(t)$. Instead, let us define $\lambda_h^c(t)$ as the stake of the honest coins that are online at time t and has been online since at least time $t-\sigma(c)$, where

$$\sigma(c)=(c-1)\left(\Delta+\frac{1+\kappa}{\lambda_{\min}}\right). \tag{2.4}$$

Here, κ is the security parameter. Intuitively, $\sigma(c)$ is a high-probability worst-case waiting delay, in seconds, of a coin for the next epoch beginning. Note that $\lambda_h^c(t)$ depends only on the stake arrival process and not on the blockchain dynamics.

The theorem below shows that the the private attack threshold yields the true security threshold:

Theorem 4. *If*

$$\frac{\lambda_h^c(t)}{1+\lambda_{\max}\Delta}>\phi_c\lambda_a(t) \qquad \textit{for all } t>0, \tag{2.5}$$

then the PoSAT generate transaction ledgers such that each transaction tx satisfies persistence (parameterized by $\tau=\rho$) and liveness (parameterized by $u=\rho$) in Definition 2 with probability at least $1-e^{-\Omega(\min\{\rho^{1-\epsilon},\kappa\})}$, for any $\epsilon>0$. The constant ϕ_c is defined in (2.22), with $\phi_1=e$ and $\phi_c\to 1$ as $c\to\infty$.

In order to prove Theorem 4, we utilize the concept of blocktree partitioning and Nakamoto blocks that were introduced in [69]. We provide a brief overview of these concepts here.

Let τ_i^h and τ_i^a be the time when the i-th honest and adversary blocks are proposed, respectively; $\tau_0^h=0$ is the time when the genesis block is proposed, which we consider as the 0-th honest block.

Definition 3. Blocktree partitioning *Given the mother tree $\mathcal{T}(t)$, define for the i-th honest block b_i, the* adversary tree $\mathcal{T}_i(t)$ *to be the sub-tree of the mother tree $\mathcal{T}(t)$ rooted at b_i and consists of all the adversary blocks that can be reached from b_i without going through another honest block. The mother tree $\mathcal{T}(t)$ is partitioned into sub-trees $\mathcal{T}_0(t)$,$\mathcal{T}_1(t)$,...$\mathcal{T}_j(t)$, where the j-th honest block is the last honest block that was proposed before time t.*

The sub-tree $\mathcal{T}_i(t)$ is born at time τ_i^h as a single block b_i and then grows each time an adversary block is appended to a chain of adversary blocks from b_i. Let $D_i(t)$ denote the depth of $\mathcal{T}_i(t)$; $D_i(\tau_i^h)=0$.

Definition 4. *[145] The j-th honest block proposed at time τ_j^h is called a* loner *if there are no other honest blocks proposed in the time interval $[\tau_j^h-\Delta,\tau_j^h+\Delta]$.*

Definition 5. *Given honest block proposal times τ_i^h's, define a honest fictitious tree $\mathcal{T}_h(t)$ as a tree which evolves as follows:*

1. *$\mathcal{T}_h(0)$ is the genesis block.*
2. *The first honest block to be proposed and all honest blocks within Δ are all appended to the genesis block at their respective proposal times to form the first level.*
3. *The next honest block to be proposed and all honest blocks proposed within time Δ of that are added to form the second level (which first level blocks are parents to which new blocks is immaterial) .*
4. *The process repeats.*

Let $D_h(t)$ be the depth of $\mathcal{T}_h(t)$.

Definition 6. *(***Nakamoto block***) Let us define:*

$$E_{ij}=\textit{event that } D_i(t)<D_h(t-\Delta)-D_h(\tau_i^h+\Delta) \textit{ for all } t>\tau_j^h+\Delta. \tag{2.6}$$

The j-th honest block is called a Nakamoto block *if it is a loner and*

$$F_j=\bigcap_{i=0}^{j-1}E_{ij} \tag{2.7}$$

occurs.

See Figure 5 in [69] for illustration of the concepts of blocktree partitioning and Nakamoto blocks.

Lemma 1. *(Theorem 3.2 in [69]) (***Nakamoto blocks stabilize***) If the j-th honest block is a Nakamoto block, then it will be in the longest chain $\mathcal{C}(t)$ for all $t > \tau_j^h + \Delta$.*

Lemma 1 states that Nakamoto blocks remain in the longest chain forever. The question is whether they exist and appear frequently regardless of the adversary strategy. If they do, then the protocol has liveness and persistence: honest transactions can enter the ledger frequently through the Nakamoto blocks, and once they enter, they remain at a fixed location in the ledger. More formally, we have the following result.

Lemma 2. *(Lemma 4.4 in [69]) Define $B_{s,s+t}$ as the event that there is no Nakamoto blocks in the time interval $[s,s+t]$ where $t \sim \Omega\left(\left[\frac{c-1}{\phi_c - 1}\right]^2\right)$. If*

$$P(B_{s,s+t}) < q_t < 1 \tag{2.8}$$

for some q_t independent of s and the adversary strategy, then the PoSAT generates transaction ledgers such that each transaction tx satisfies persistence *(parameterized by $\tau = \rho$) and* liveness *(parameterized by $u = \rho$) in Definition 2 with probability at least $1 - q_\rho$.*

In order to prove Lemma 2, we proceed in six steps as illustrated in Fig. 2-6.

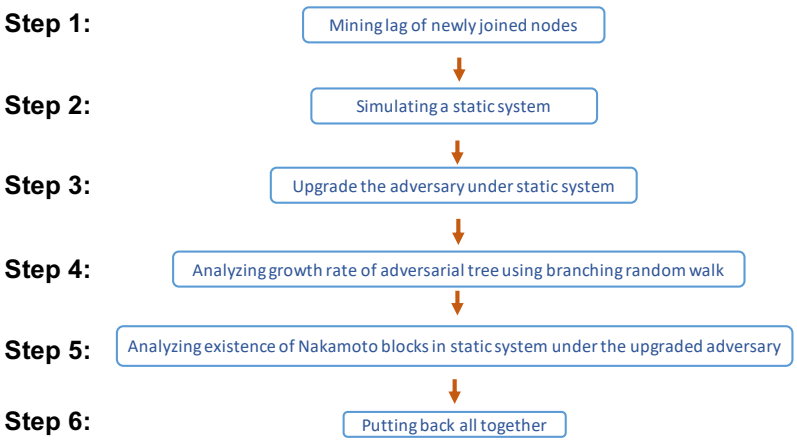


Figure 2-6: Flowchart of the proof for Lemma 2.

2.4.2 Step 1: Mining lag of newly joined nodes

From section 5.2, recall that $\lambda_h(t)$ is defined as the stake of the coins that are online at time t but has encountered at least one epoch beginning. That implies, within an epoch, $\lambda_h(t)$ is the effective honest stake that can be used to contribute towards the growth of the longest chain; it remains constant and gets updated only at the epoch beginning. In order to analyze the effect of this lag in a honest node to start mining, we simulate a new dynamic available system, $dyn2$, where, at time t, an honest coin can contribute towards the growth of the longest chain if it has been online in the original dynamic system since at least time $t - \sigma(c)$, where, $\sigma(c) > 0$. Recall that $\lambda_h^c(t)$ be defined as the stake of the coins that are online at time t in the original dynamic system and has been online since at least $t - \sigma(c)$. Clearly, $\lambda_h^c(t)$ is the rate at which the honest nodes win leader election at time t in $dyn2$. We have the following relationship between the original dynamic available system and $dyn2$.

Lemma 3. *For the dynamic available system $dyn2$ and for all $s,t > 0$, define $B_{s,s+t}^{dyn2}$ as the event that there are no Nakamoto blocks in the time interval $[s, s+t]$. Let κ_0 be the solution for the equation*

$\ln\left(\frac{\lambda_{\max}}{\lambda_{\min}}(1+\kappa)\right)=\kappa$. *Then, for* $\sigma(c)=(c-1)\left(\Delta+\frac{1+\kappa}{\lambda_{\min}}\right)$ *and* $\kappa>>\kappa_0$, *we have*

$$P(B_{s,s+t})\leq P(B^{dyn2}_{s,s+t})+e^{-\Omega(\kappa)}.$$

The proof is given in Appendix 2.8.1.

2.4.3 Step 2: Simulating a static system

Without loss of generality, we assume that the adversarial power is boosted such that $\lambda_a(t)$ satisfies (2.5) with equality for all t. Let us define η such that $\lambda_a(t)=(1-\eta)\lambda_h(t)$ for all t. Let λ_h be some positive constant. Taking $dyn2$ as the base, we simulate a static system, $ss0$, where both honest nodes and adversary win leader elections with constant rates λ_h and λ_a satisfying $\lambda_a=(1-\eta)\lambda_h$. This requires, for a local time $t>0$ in $dyn2$, defining a new local time $\alpha(t)$ for $ss0$ such that

$$\lambda^c_h(u)du=\lambda_h d\alpha \Longrightarrow \alpha(t)=\int_0^t \frac{\lambda^c_h(u)}{\lambda_h}du. \tag{2.9}$$

Additionally, for every arrival of an honest or adversarial block in $dyn2$ at a particular level at a tree, there is a corresponding arrival in $ss0$ at the same level in the same tree. For a time t in the local clock of $dyn2$, let $\Delta^{ss0}(t)$ be the network delay of $dyn2$ measured with reference to the local clock of $ss0$. Using (2.9), we have

$$\frac{\lambda_{\min}}{\lambda_h}\Delta\leq\Delta^{ss0}(t)\leq\frac{\lambda_{\max}}{\lambda_h}\Delta. \tag{2.10}$$

We have the following relationship between $dyn2$ and $ss0$.

Lemma 4. *Consider the time interval* $[s,s+t]$ *in the local clock of* $dyn2$. *For the static system* $ss0$, *define* $B^{ss0}_{\alpha(s),\alpha(s+t)}$ *as the event that there are no Nakamoto blocks in the time interval* $[\alpha(s),\alpha(s+t)]$ *in the local clock of* $ss0$. *Then,*

$$P(B^{dyn2}_{s,s+t})=P(B^{ss0}_{\alpha(s),\alpha(s+t)}).$$

The proof for this lemma is given in Appendix 2.8.2.

2.4.4 Step 3: Upgrading the adversary

As the occurrence of Nakamoto blocks is a race between the fictitious honest tree and the adversarial trees from the previous honest blocks, we next turn to an analysis of the growth rate of an adversary tree. However, the growth rate of an adversarial tree would now depend on the location of the root honest block within an epoch which adds to the complexity of the analysis. To get around this complexity, we simulate a new static system, $ss1$ in which the adversary, on winning a leader election after evaluating RandVDF.Eval and appending a block to an honest block (that is, growing a new adversarial tree), is given a gift of chain of $c-1$ extra blocks for which the adversary doesn't have to compute RandVDF.Eval. Thus, the adversary has to compute only one RandVDF.Eval for the chain of first c blocks in the adversarial tree. At this point, the adversary can assume a new epoch beginning and accordingly update `randSource`. Hereafter, the evolution of `randSource` follows the rules in $ss0$. Note that the local clock for both the static systems $ss0$ and $ss1$ are same. Now, we have the following relationship between $ss0$ and $ss1$.

Lemma 5. *Consider the time interval* $[s,s+t]$ *in the local clock of* $dyn2$. *For the static system* $ss1$, *define* $B^{ss1}_{\alpha(s),\alpha(s+t)}$ *as the event that there are no Nakamoto blocks in the time interval* $[\alpha(s),\alpha(s+t)]$ *in the local clock of* $ss1$. *Then,*

$$P(B^{ss0}_{\alpha(s),\alpha(s+t)})\leq P(B^{ss1}_{\alpha(s),\alpha(s+t)}).$$

The proof for this lemma is given in Appendix 2.8.3.

For analyzing $P(B^{ss1}_{\alpha(s),\alpha(s+t)})$, we first consider an arbitrary static system $ss2$ where both honest nodes and adversary win leader elections with constant rates λ_h and λ_a, respectively, the honest nodes follows PoSAT , the adversary has similar additional power of gift of chain of $c-1$ blocks as in $ss1$ but the network delay is a constant, say Δ'. For some $s',t'>0$ in the local clock of the static system $ss2$, we will determine an upper bound on $P(B^{ss2}_{s',s'+t'})$ in Sections 2.4.5 - 2.4.6 and then use this result to obtain an upper bound on $P(B^{ss1}_{\alpha(s),\alpha(s+t)})$ in Section 2.4.7.

2.4.5 Step 4: Growth rate of the adversarial tree

For time $t'>0$, let $\hat{\mathcal{T}}_i(t')$ represents the adversarial tree in $ss2$ with i^{th} honest block as its root. The depth $D_i(t')$ at time t' in the local clock of $ss2$ is defined as the maximum depth of the blocks of $\hat{\mathcal{T}}_i(t')$ at time t'. In Lemma 6, we evaluate the tail bound on $D_i(t')$.

Lemma 6. *For $x>0$ so that $\eta_c\lambda_a t'+x$ is an integer,*

$$P(D_i(t')\geq \phi_c\lambda_a t'+cx)\leq e^{-\theta^*_c t'}e^{(\eta_c\lambda_a t'+x-1)\Lambda_c(\theta^*_c)}g(t'). \tag{2.11}$$

where $\phi_c=c\eta_c$, $g(t')=\sum_{i_1\geq 1}\int_0^{t'}\frac{\lambda_a^{i_1}u^{i_1-1}e^{-\lambda_a u}}{\Gamma(i_1)}e^{\theta^*_c u}du$, $\Lambda_c(\theta_c)=\log(-\lambda_a^c/\theta_c(\lambda_a-\theta_c)^{c-1})$ and θ^*_c is the solution for the equation $\Lambda_c(\theta)=\theta\dot{\Lambda}_c(\theta)$

Details on the analysis of $\hat{\mathcal{T}}_i(t')$ and the proof of Lemma 6 are in Appendix 2.8.4.

2.4.6 Step 5: Existence of Nakamoto blocks

With Lemma 6, we show below that in the static system $ss2$ in the regime $\phi_c\lambda_a<\frac{\lambda_h}{1+\lambda_h\Delta'}$, Nakamoto blocks has a non-zero probability of occurrence.

Lemma 7. *If*

$$\phi_c\lambda_a<\frac{\lambda_h}{1+\lambda_h\Delta'},$$

then, in the static system $ss2$, there is a $p>0$ such that the probability of the $j-th$ honest block being a Nakamoto block is greater than p for all j.

The proof of this result can be found in Appendix 2.8.5.

Having established the fact that Nakamoto blocks occurs with non-zero frequency, we can bootstrap on Lemma 7 to get a bound on the probability that in a time interval $[s',s'+t']$, there are no Nakamoto blocks, i.e. a bound on $P(B_{s',s'+t'})$.

Lemma 8. *If*

$$\phi_c\lambda_a<\frac{\lambda_h}{1+\lambda_h\Delta'},$$

then for any $\epsilon>0$, there exist constants $\bar{a}_\epsilon,\bar{A}_\epsilon$ so that for all $s'\geq 0$ and $t'>\max\left\{\left(\frac{2\lambda_h}{1-\eta}\right)^2\left(\frac{c-1}{\phi_c-1}\right)^2,\left[(c-1)\left(\Delta'+\frac{1}{\lambda_{\min}}\right)\right]^2\right\}$, we have

$$P(B^{ss2}_{s',s'+t'})\leq\bar{A}_\epsilon\exp(-\bar{a}_\epsilon t'^{1-\epsilon}) \tag{2.12}$$

where $\bar{a}_\epsilon$ is a function of Δ'.

The proof of this result can be found in Appendix 2.8.6.

2.4.7 Step 6: Putting back all together

In this section, we use the results from Section 2.4.6 to upper bound $P(B^{ss1}_{\alpha(s),\alpha(s+t)})$ and hence, $P(B_{s,s+t})$.

Using equation 2.9, we have $\phi_c\lambda_a(t)<\frac{\lambda^c_h(t)}{1+\lambda_{\max}\Delta}\iff\phi_c\lambda_a<\frac{\lambda_h}{1+\lambda_{\max}\Delta}$. Then, we have the following lemma:

Lemma 9. *If*

$$\phi_c \lambda_a(t) < \frac{\lambda_h^c(t)}{1+\lambda_{\max}\Delta},$$

then for any $\epsilon > 0$ *there exist constants* $\bar{a}_\epsilon, \bar{A}_\epsilon$ *so that for all* $s \geq 0$ *and* $t > \max\left\{ \left(\frac{2\lambda_h}{1-\eta}\right)^2 \left(\frac{\lambda_h}{\lambda_{\min}}\right) \left(\frac{c-1}{\phi_c - 1}\right)^2, \left(\frac{\lambda_h}{\lambda_{\min}}\right) \left[(c-1)\left(\Delta + \frac{1}{\lambda_{\min}}\right)\right]^2 \right\}$, *we have*

$$P(B_{s,s+t}) \leq \bar{A}_\epsilon \exp(-\bar{a}_\epsilon t^{1-\epsilon}) + e^{-\Omega(\kappa)}. \tag{2.13}$$

The proof for this result is given in Appendix 2.8.7. Then, combining Lemma 9 with Lemma 2 implies Theorem 4.

2.5 Discussion

In this section, we discuss some of the practical considerations in adopting PoSAT .

A key question in PoSAT is what is the right choice of c? If c is low, say 10, then the security threshold is approximately 1.58. At $c = 10$, the protocol is fully unpredictable and the confirmation latency is not too high. Also, any newly joining honest node has to wait for around 10 inter-block arrivals before it can participate in leader election. Thus, if there is a block arrival every second, then, the node has to wait for 10 secs. In any standard blockchain, there is always a bootstrap period for the node to ensure that the state is synchronized with the existing peers and 10 secs is negligible as compared to the bootstrap period.

In PoSAT , a separate RandVDF needs to be run for each public-key. In a purely decentralized implementation, all nodes may not have the same rate of computing VDF. This may disadvantage nodes whose rate of doing sequential computation is slower. One approach to solve this problem is to build open-source hardware for VDF - this is already under way through the VDF Alliance. Even under such a circumstnace, it is to be expected that nodes that can operate their hardware in idealized circumstances (for example, using specialized cooling equipment) can gain an advantage. A desirable feature of our protocol is that gains obtained by a slight advantage in the VDF computation rate are bounded. For PoSAT , a combination of the VDF computation rate and the stake together yields the net power weilded by a node, and as long as a majority of such power is controlled by honest nodes, we can expect the protocol to be safe.

In our PoSAT specification, the difficulty parameter for the computation of RandVDF.Eval was assumed to be fixed. This threshold was chosen based on the entire stake being online - this was to ensure that forking even when all nodes are present remains small, i.e., $\lambda_{\max}\Delta$ remained small. In periods when far fewer nodes are online, this leads to a slowdown in confirmation latency. A natural way to mitigate this problem is to use a variable mining threshold based on past history, similar to the adaptation inherent in Bitcoin. A formal analysis of Bitcoin with variable difficulty was carried out in [90, 92], we leave a similar analysis of our protocol for future work.

In our protocol statement, we have used the RandVDF directly on the randomness prevRand and the public key. The RandVDF ensures that any other node can only predict a given node's leadership slot at the instant that it actually wins the VDF lottery. However, this still enables an adversary to predict the leadership slots of nodes that are offline and can potentially bribe them to come online to favor the adversary. In order to eliminate this exposure, we can replace the hash in the mining condition by using a verifiable random function [124, 70] (which is calculated using the node's secret key but can be checked using the public key). This ensures that an adversary which is aware of all the public state as well as private state of all *online* nodes (including their VRF outputs) still cannot predict the leadership slot of any node ahead of the time at which they can mine the block. This is because, such an adversary does not have access to the VRF output of the offline nodes.

There are two types of PoS protocols: one favoring liveness under dynamic availability and other favoring safety under asynchrony. BFT protocols fall into the latter class and lack dynamic availability. One shortcoming of the longest chain protocol considered in the paper is the reduced throughput and latency compared to the fundamental limits; this problem is inherited from the Nakamoto consensus for

PoW [127]. However, a recent set of papers address these problems in PoW (refer Prism [27], OHIE [165] and Ledger Combiners [81]). Adaptations of these ideas to the PoSAT protocol is left for future work. Furthermore, our protocol, like Nakamoto, does not achieve optimal chain quality. Adopting ideas from PoW protocols with optimal chain quality, such as Fruitchains [133], is also left for future work.

Finally, while we specified PoSAT in the context of proof-of-stake, the ideas can apply to other mining modalities - the most natural example is proof-of-space. We note that existing proof-of-space protocols like Chia [56], use a VDF for a fixed time, thus making the proof-of-space challenge predictable. In proof-of-space, if the predictability window is large, it is possible to use slow-storage mechanisms such as magnetic disks (which are asymmetrically available with large corporations) to answer the proof-of-space challenges. Our solution of using a RandVDF can be naturally adapted to this setting, yielding unpredictability as well as full dynamic availability.

2.6 Suite of Possible Attacks Under Dynamic Availability

In this section, we describe the suite of possible attacks under dynamic availability in PoS systems. This attacks are possible even under static stake. We also discuss some design recommendations for mitigating against such attacks in PoS systems.

2.6.1 Content-grinding attack

Referring to Fig 2-4, we note that the content of the block, namely the transactions, were not used in determining whether the THRESHOLD(s) is satisfied or not. If we instead checked whether HASH(O_i,`slot`,`transactionList`) < THRESHOLD(s) instead of HASH(O_i,`slot`) < THRESHOLD(s) the protocol loses security due to the ability of adversary to choose the set of transactions in order to increase its likelihood of winning the leadership certificate.

In such a case, the adversary can get unlimited advantage by performing such content-grinding by parallel computation over different sets and orders of transactions. We note that in PoW the adversary does not gain any advantage by performing such content grinding, since it is equivalent to grinding on the Nonce, which is the expected behavior anyway.

2.6.2 Sybil attack

One natural attack in PoS for an adversary to sybil the stake contained in a single coin and distribute it across multiple coins which might increase the probability of winning a leader election from at least one of the coins. We describe next that having difficulty parameter in RANDVDF.EVAL proportional to the stake of the coin defends against such sybil attack. Let us consider H to be the value of the hash function on the output of VDF in an iteration, R to be range of this hash function and th be the difficulty parameter that is proportional to the stake of the coin. Suppose $p = P(\mathrm{H} < th) = \frac{th}{R}$, which is the probability of winning the leader election in each iteration of the VDF. If we sybil the stake into , let's say, three coins with equal stakes, then, the probability of winning leader election for each individual stake in each iteration of VDF is $P(\mathrm{H} < \frac{th}{3}) = \frac{th}{3R} = \frac{p}{3}$. This is due to the fact that difficulty parameter th is proportional to the stake. Hence, the probability of winning leader election in each iteration of VDF by at least one coin is $1-\left(1-\frac{p}{3}\right)^3$. However, as the VDFs are iterating very fast, we are in the regime $0 < p << 1$. Thus, by Binomial series expansion, $1-\left(1-\frac{p}{3}\right)^3 = 1-(1-3\frac{p}{3}+O(p^2)) = p+O(p^2)$. Hence, this validates our design choice that difficulty parameter is proportional to the stake of the coin. This design choice is not unique to our design and is common in all longest chain based proof-of-stake protocols.

2.6.3 Costless simulation attack

Both sleepy model of consensus [136] and Ouroboros Genesis [26] have a weaker definition of dynamic availability: *all adversary nodes are always online starting from genesis and no new adversary nodes can join*, which makes them vulnerable to costless simulation attack as described next. In case of sleepy model of consensus [136], as shown in Fig 2-7, suppose that in the 1^{st} year of the existence of the PoS

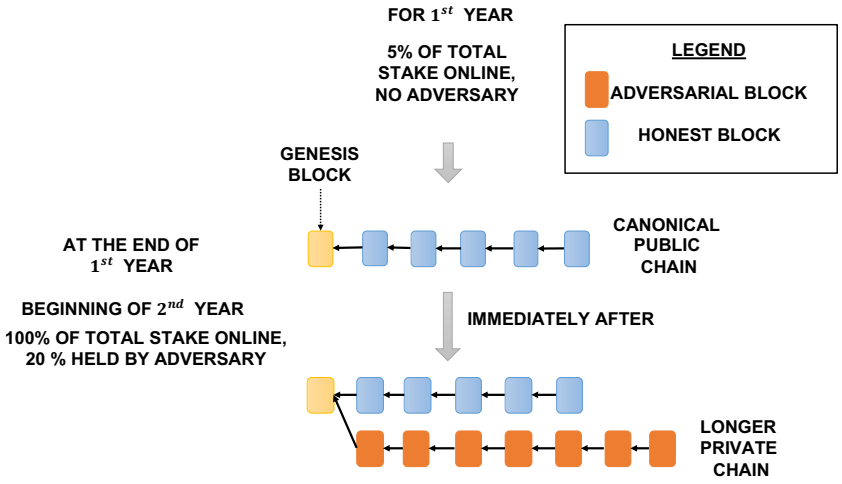


Figure 2-7: Costless simulation attack for sleepy model of consensus [136].

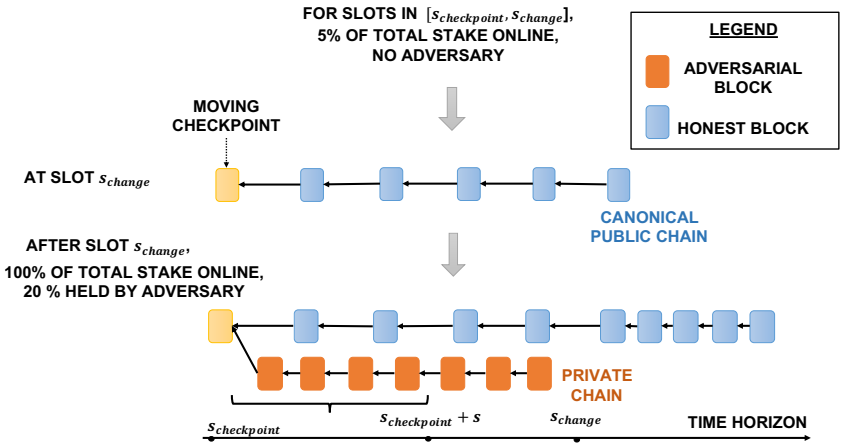


Figure 2-8: Costless simulation attack for Ouroboros Genesis [26].

system, only 5% of the total stake, all honest, is online and actively participating in evolution of the blockchain. Consider that at the beginning of the 2^{nd} year, all 100% of the stake is online with 20% of the stake being controlled by the adversary. The adversary can costlessly simulate (with requirement of little computational time) the eligibility condition in sleepy protocol across large range of values of time t, thus, constructing a longer private chain than the canonical public chain. In sleepy, under the fork-choice rule of choosing the longest chain, the private chain will be selected as the canonical chain once it is revealed by the adversary. In case of Ouroboros Genesis [26], as shown in Fig 2-8, the adversary can utilize the 20% stake under its control after the slot s_{change} to costlessly construct a private chain involving leader elections for the slots starting from the checkpoint slot $s_{checkpoint}$. Observe that in the slots $[s_{checkpoint}, s_{change}]$ of the operation of the PoS system, the canonical public chain evolved due to the participation of only 5% stake. Clearly, with high probability, for any s such that $s_{checkpoint} + s < s_{change}$, the private chain has more blocks in the range $[s_{checkpoint}, s_{checkpoint} + s]$ as compared to canonical public chain. Under the fork-choice rule of Ouroboros Genesis as described in Fig. 10 of [26], the private adversarial chain will be selected as the canonical chain when it is revealed.

If the fork-choice rule is to choose the longest chain, the design recommendation for defending against costless simulation attack is to make it *expensive* for the adversary to propose blocks for the past slots and create longer chain. For instance, in PoSAT , the adversary would have to initiate its RandVDF from the first block of the epoch where `randSource` is updated. Due to the sequential

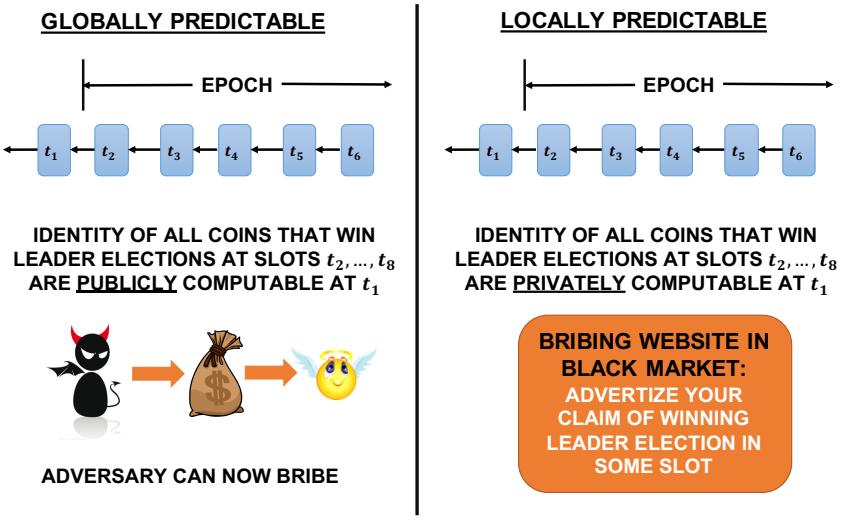


Figure 2-9: Variations of bribery attack stemming from predictability.

nature of the computation of RandVDF, with high probability, the adversary won't be able to create a private chain longer than the canonical chain.

2.6.4 Bribery attack due to predictability

A key property of PoW protocols is their ability to be unpredictable: no node (including itself) can know when a given node will be allowed to propose a block ahead of the proposal slot. In the existing PoS protocols, there are two notions of predictability depending on how the leader election winner is decided - *globally predictable* and *locally predictable*. Referring to Fig 2-9, using hash function for deciding winner of leader election, as in [136], renders the identity of winners of leader elections in future slots publicly computable. An adversary can now bribe a coin that is going to propose a block in a future slot to include or exclude a specific transaction of adversary's choice or influence the position on where to append the block. On the other hand, using verifiable random function (VRF) for deciding winner of leader election, as in Ouroboros Praos [65], Ouroboros Genesis [26] and Snow White [32], mutes the aforementioned public computability. However, a node owning a coin can still locally compute the future slots in which that coin can win the leader election. Now, the node can advertise its future electability in the black market.

The central idea on how to avoid such predictability is to ensure that a node owning a coin shouldn't learn about winning a leader election with that coin in slot s before the slot s. In PoSAT , owing to randomness of `randIter` in RandVDF, the node learns about winning a leader election for that coin in slot s only after completion of sequential execution of the RandVDF at slot s.

2.6.5 Private attack by enumerating blocks within an epoch

In PoSAT , at the beginning of each epoch, the `randSource` is updated. However, if the appropriate guardrail in the form of time-ordering (line 38 in Algorithm 2) is not put into place, then, this `randSource` update can provide statistical advantage to the adversary in creating longer private chain. To be specific, suppose that PoSAT doesn't require the `slot` in the blocks of a chain to be ordered in the ascending order. Then, as shown in Fig 2-10, the adversary can enumerate over the c blocks in the private adversarial tree to have c different `randSource` updates for the next epoch. This gives c distinct opportunities to the adversary to evolve the private adversarial tree which gives the aforementioned statistical advantage of order c in terms of inter-arrival time of the adversarial blocks. With the guardrail of time-ordering in place, as in PoSAT , the aforementioned enumeration is not possible as the `slot` contained in the blocks of a chain are required to be in ascending order.

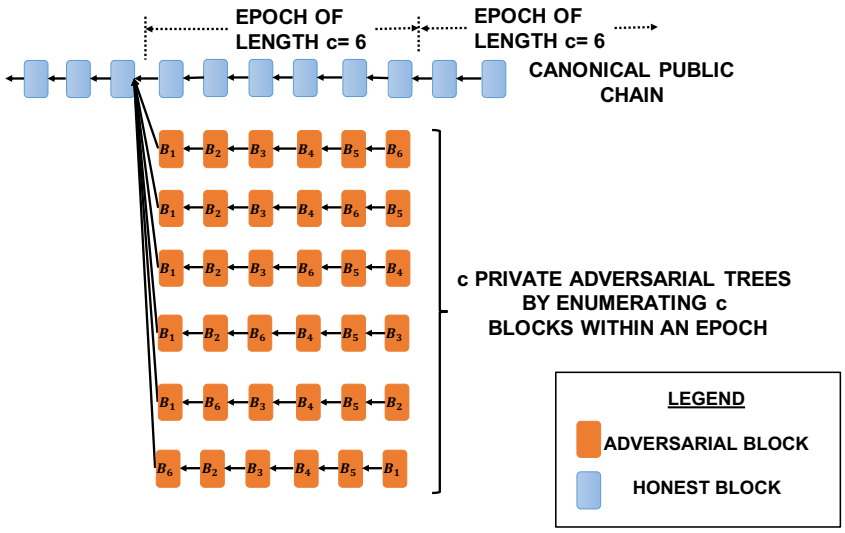


Figure 2-10: Enumerating blocks when time-ordering is not required.

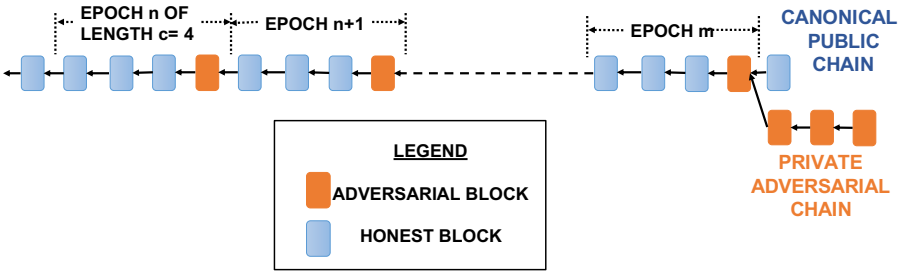


Figure 2-11: Illustration of the long-range attack. Consider that $m > n$.

2.6.6 Long-range attack by leveraging randomness update

Updating `randSource` for a new epoch based on solely the last block of the previous epoch, as done in PoSAT , gives rise to an unique situation in which an adversary can mount a long-range attack to create a longer private adversarial chain. Referring to Fig 2-11, an adversary, with sufficiently large probability, can win at least one leader election in each epoch and publicly reveal the block associated with that leader election after appropriate delay so that the block ends up as the last block of that epoch. Observe that this strategy will give power to the adversary to dictate the `randSource` for each epoch. Moreover, the adversary, on proposing on winning a leader election for an epoch, can just move on to contesting a leader election for the next epoch. Thus, the adversary can privately behave as if $c = 1$ whereas the actual c might be greater than 1. With such a strategy, the adversary can win at least one leader election for many future epochs and publicly reveal the blocks associated with those leader elections in a time-appropriate manner. The adversary can continue this strategy until an appropriate epoch when it is able to win multiple leader elections and wants to do double spending. There are two design recommendations on how to protect against this long-range attack:

- requiring time-ordering of the blocks in a chain, as done in PoSAT , would ensure that even after the adversary behaves as if $c = 1$ and wins leader elections for future epochs, the blocks associated with those leader elections would fail the time-ordering (line 38 in Algorithm 2). This completely removes the aforementioned long-range attack.
- requiring that the `randSource` for a new epoch is dependent on all the blocks of the previous epoch. This strategy diminishes the amount of influence that an adversary can have on the `randSource` update.

2.7 Supplementary for Section 2.2.1

We give a brief description of VDFs, starting with its definition.

Definition 7 (from [36]). *A VDF $V=(Setup,Eval,Verify)$ is a triple of algorithms as follows:*

- *$Setup(\lambda,\tau)\to \mathbf{pp}=(ek,vk)$ is a randomized algorithm that takes a security parameter λ and a desired puzzle difficulty τ and produces public parameters $\mathbf{pp}$ that consists of an evaluation key ek and a verification key vk. We require Setup to be polynomial-time in λ. By convention, the public parameters specify an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. We assume that $\mathcal{X}$ is efficiently sampleable. Setup might need secret randomness, leading to a scheme requiring a trusted setup. For meaningful security, the puzzle difficulty τ is restricted to be sub-exponentially sized in λ.*

- *$Eval(ek,input,\tau)\to(O,proof)$ takes an input $\in\mathcal{X}$ and produces an output $O\in\mathcal{Y}$ and a (possibly empty) $proof$. Eval may use random bits to generate the $proof$ but not to compute O. For all $\mathbf{pp}$ generated by $Setup(\lambda,\tau)$ and all input $\in\mathcal{X}$, algorithm $Eval(ek,input,\tau)$ must run in parallel time τ with $poly(log(\tau),\lambda)$ processors.*

- *$Verify(vk,input,O,proof)\to Yes,No$ is a deterministic algorithm takes an input, output and proof and outputs Yes or No. Algorithm Verify must run in total time polynomial in $log\tau$ and λ. Notice that $Verify$ is much faster than $Eval$.*

The definition for correctness and soundness for RandVDF is defined as follows:

Definition 8 (Correctness). *A RandVDF V is correct if for all λ,τ, parameters $(ek,vk)\xleftarrow{\$}\textsc{Setup}(\lambda)$, and all input $\in X$, if $(O,proof)\xleftarrow{\$}\textsc{Eval}(ek,input,\tau)$ then $\textsc{Verify}(vk,input,O,proof)=Yes$.*

Definition 9 (Soundness). *A RandVDF is sound if for all algorithms $\mathcal{A}$ that run in time $O(poly(t,\lambda))$*

$$Pr\left[\begin{matrix}\textsc{Verify}(vk,input,O,proof)=Yes\\ O\neq\textsc{Eval}(ek,input,\tau)\end{matrix}\middle|\begin{matrix}pp=(ek,vk)\xleftarrow{\$}\textsc{Setup}(\lambda)\\ (input,O,proof)\xleftarrow{\$}\mathcal{A}(\lambda,pp,\tau)\end{matrix}\right]\leq negl(\lambda)$$

2.8 Proofs

2.8.1 Proof of Lemma 3

First, we prove the following lemma.

Lemma 10. *Define*

$$E_1=\{\text{There is no epoch-beginning within the time interval } [t-\sigma(c),t]\} \tag{2.14}$$

and, let κ_0 be the solution for the equation $\ln\left(\frac{\lambda_{\max}}{\lambda_{\min}}(1+\kappa)\right)=\kappa$. Then, for $\sigma(c)=(c-1)\left(\Delta+\frac{1+\kappa}{\lambda_{\min}}\right)$ and $\kappa>>\kappa_0$, we have

$$P(E_1)\leq e^{-O(\kappa)}.$$

Proof. Define $X_d,d>0$, as the time it takes for D_h in the original dynamic available system to reach depth d after reaching depth $d-1$. Then, for some $d_0>0$, we have $E_1=\left\{\sum_{d=d_0}^{d_0+c-2}X_d>\sigma(c)\right\}$. Observe that, due to our blocktree partitioning, $X_d=\Delta+Y_d$, where Y_d is a non-homogeneous exponential

random variable. Therefore, by Chernoff bound, for any $v>0$

$$\begin{aligned}
P\left(\sum_{d=d_0}^{d_0+c-2} X_d > \sigma(c)\right) &\leq \mathbb{E}\left(e^{v\sum_{d=d_0}^{d_0+c-2} X_d - v\sigma(c)}\right) \\
&= \mathbb{E}\left(e^{v\sum_{d=d_0}^{d_0+c-2} Y_d}\right) e^{v(c-1)\Delta - v\sigma(c)} \\
&\overset{(a)}{=} e^{v(c-1)\Delta - v\sigma(c)} \mathbb{E}_{Y_{d_0}|X_{d_0-1}} e^{vY_{d_0}} \cdots \mathbb{E}_{Y_{d_0+c-2}|X_{d_0-1}\cdots X_{d_0+c-3}} e^{vY_{d_0+c-2}} \\
&\overset{(b)}{\leq} e^{v(c-1)\Delta - v\sigma(c)} \left(\frac{\lambda_{\max}}{\lambda_{\min} - v}\right)^{c-1} \\
&\overset{(c)}{=} e^{-v\frac{(c-1)(1+\kappa)}{\lambda_{\min}}} \left(\frac{\lambda_{\max}}{\lambda_{\min} - v}\right)^{c-1}
\end{aligned}$$

where (a) is due to law of total expectation, (b) is due to the fact that, if $f_{Y_{d_0+i}|X_{d_0-1}\cdots X_{d_0+i-1}}(y)$ is the pdf of Y_{d_0+i} given $X_{d_0-1}\cdots X_{d_0+i-1}$, then $\lambda_{\min} \leq \lambda_h(t) \leq \lambda_{\max}$ implies $f_{Y_{d_0+i}|X_{d_0-1}\cdots X_{d_0+i-1}}(y) \leq \lambda_{\max} e^{-\lambda_{\min} y}$, (c) is by putting $\sigma(c) = (c-1)\left(\Delta + \frac{1+\kappa}{\lambda_{\min}}\right)$. Optimizing over v implies that for $v = \lambda_{\min}\left(\frac{\kappa}{1+\kappa}\right)$, we have

$$P\left(\sum_{d=d_0}^{d_0+c-2} X_d > \sigma(c)\right) \leq e^{(c-1)\left[-\kappa + \ln\left(\frac{\lambda_{\max}}{\lambda_{\min}}(1+\kappa)\right)\right]}$$

Note that for $\kappa >> \kappa_0$, we have $P\left(\sum_{d=d_0}^{d_0+c-2} X_d > \sigma(c)\right) \leq e^{-O(\kappa)}$. □

Recall that, under the design of simulated system *dyn*2, if an honest coin has been online in the original dynamic available system for at least time $\sigma(c)$, then the coin can also contribute to the growth of the canonical chain in *dyn*2. From Lemma 10, we know that for $\sigma(c) = (c-1)\left(\Delta + \frac{1+\kappa}{\lambda_{\min}}\right)$ and $\kappa >> \kappa_0$, this honest coin has encountered at least one epoch-beginning in the original dynamic available system with probability $1-e^{-O(\kappa)}$. That implies, with high probability $1-e^{-O(\kappa)}$, at time t, if an honest coin is contributing to the growth of the canonical chain in *dyn*2, then it is also contributing to the growth of the canonical chain in the original dynamical system. However, observe that at the same time t in the original dynamic available system, there might be other honest coins which became online after $t-\sigma(c)$ and have encountered at least one epoch-beginning. At time t, these coins would contribute to the growth of the canonical chain in the original dynamic available system but won't be contributing to the growth of the canonical chain in *dyn*2. Thus, $\lambda_h^c(t) \leq \lambda_h(t)$ with probability $1-e^{-O(\kappa)}$. Consequently, $P(B_{s,s+t}) \leq P(B_{s,s+t}^{dyn2}) + e^{-O(\kappa)}$.

2.8.2 Proof of Lemma 4

Observe that from (2.9), we have

$$\int_{t_1}^{t_2} \lambda_h^c(t)dt = \lambda_h[\alpha(t_2) - \alpha(t_1)] \tag{2.15}$$

Thus, $\alpha(t)$ is an increasing function in t. Then, we have the following lemma.

Lemma 11. *The ordering of events in the dynamic available system dyn*2 *is same as in the static system ss*0*.*

Proof. Suppose there are two events E_1 and E_2 that happen in *dyn*2 such that $t_{E_1} < t_{E_2}$, that is, E_1 happen before E_2 in *dyn*2. By contradiction, assume that E_2 happen before E_1 in the frame of

reference of the static system $ss0$. By equation 2.9, that implies, $\alpha(t_{E_2}) < \alpha(t_{E_1})$. However, this contradicts the fact that $\alpha(t)$ is an increasing function in t. □

Suppose that $B_{s,s+t}$ happens in $dyn2$. This implies that, in $dyn2$, for every honest block b_j proposed at $\tau_j^h \in [s,s+t]$, there exists some minimum time $t_0 > \tau_j^h + \Delta$ and some honest block b_i proposed at τ_i^h such that

$$D_i(t_0) \geq D_h(t_0 - \Delta) - D_h(\tau_i^h + \Delta).$$

Due to Lemma 11, events in the evolution of the blockchain in $dyn2$ during the interval $[\tau_i^h, t_0]$ happens in the same order in the static system $ss0$ during the time interval $[\alpha(\tau_i^h), \alpha(t_0)]$. That implies the depth of the fictitious honest tree at time t in the local clock of $dyn2$ is same as the depth of the same fictitious honest tree at time $\alpha(t)$ in the local clock of $ss0$. This equivalence also carries over for the adversarial trees. Then, analysing the race between the fictitious honest tree $\mathcal{T}_h(t)$ and the adversarial tree $\mathcal{T}_i(t)$ with respect to the local clock of $ss0$, we can write

$$D_i(\alpha(t_0)) \geq D_h(\alpha(t_0 - \Delta)) - D_h(\alpha(\tau_i^h + \Delta))$$

That implies b_j is not a Nakamoto block in the static system $ss0$ too. Since, b_j is any arbitrary honest block with $\tau_j^h \in [s, s+t]$, therefore this is true for all honest blocks j' with $\tau_{j'}^h \in \{s, s+t\}$. Hence, $B_{s,s+t}^{dyn2} = B_{\alpha(s),\alpha(s+t)}^{ss0}$ which implies $P(B_{s,s+t}^{dyn2}) = P(B_{\alpha(s),\alpha(s+t)}^{ss0})$. This concludes our lemma.

2.8.3 Proof of Lemma 5

For simulating the static system $ss1$, keep the sample path of the progress of the fictitious honest tree in both static systems $ss0$ and $ss1$ same. For some $t > 0$ in the local clock of $dyn2$, let $\mathcal{T}_i(t)$ represent the adversarial tree in $ss0$ with b_i as its root. Suppose $B_{\alpha(s),\alpha(s+t)}^{ss0}$ happens in $ss0$ for some $s, t > 0$ defined in the local clock of $dyn2$. That implies, for every honest block b_j proposed at $\alpha(\tau_j^h) \in [\alpha(s), \alpha(s+t)]$, there exists some minimum time $\alpha(t_0) > \alpha(\tau_j^h + \Delta)$ and some honest block b_i proposed at $\alpha(\tau_i^h)$ such that

$$D_i(\alpha(t_0)) \geq D_h(\alpha(t_0 - \Delta)) - D_h(\alpha(\tau_i^h + \Delta)).$$

Now, for any arbitrary b_j, there are two cases:

1. If the tip of the fictitious honest tree at time $\alpha(t_0 - \Delta)$ is in the same epoch as the honest block b_i, then, the adversary can duplicate the first block in the adversarial tree $\mathcal{T}_i(t)$ of the static system $ss0$ and attach it to the block b_i of the simulated system $ss1$. However, in $ss1$, the adversary immediately gets a gift of $c-1$ blocks.

2. If the fictitious honest tree at time $\alpha(t_0 - \Delta)$ is in a different epoch as the honest block b_i, then, the adversary can duplicate the $\mathcal{T}_i(t)$ and prune it to contain all the blocks starting from the epoch that comes immediately after the epoch containing b_i in $\mathcal{T}_i(t)$. Then, in $ss1$, the adversary duplicates the first block in the adversarial tree $\mathcal{T}_i(t)$ of the static system $ss0$ and attaches it to the block b_i of the simulated system $ss1$ that immediately gifts a chain of $c-1$ blocks. The adversary then appends over that chain the pruned $\mathcal{T}_i(t)$.

Both cases clearly imply that at time $\alpha(t_0)$, there is an adversarial tree on b_i in $ss1$ whose depth is greater than $\mathcal{T}_i(t_0)$ in $ss0$. Thus, b_j is not a Nakamoto block in $ss1$. Hence, $P(B_{\alpha(s),\alpha(s+t)}^{ss0}) \leq P(B_{\alpha(s),\alpha(s+t)}^{ss1})$.

2.8.4 Growth rate of Adversarial Tree $\hat{\mathcal{T}}_i(t)$

We first give a description of the (dual of the) adversarial tree consisting of super-blocks in terms of a Branching Random Walk (BRW).

Observe that due to the assumption on adversary in $ss2$, each adversarial tree $\hat{\mathcal{T}}_i(t')$ (with i^{th} honest block as its root), when analysed in the local clock of $ss2$, grows statistically in the same way, without

any dependence on the level of the root. Without loss of generality, let us focus on the adversary tree $\hat{\mathcal{T}}_0(t')$, rooted at genesis. The genesis block is always at depth 0 and hence $\hat{\mathcal{T}}_0(0)$ has depth zero.

We can transform the tree $\hat{\mathcal{T}}_0(t')$ into a new random tree $\hat{\mathcal{T}}_0^s(t')$. Every c generations in $\hat{\mathcal{T}}_0(t')$ can be viewed as a single generation in $\hat{\mathcal{T}}_0^s(t')$. Thus, every block in $\hat{\mathcal{T}}_0^s(t')$, termed as *superblocks*, is representative of c blocks in $\hat{\mathcal{T}}_0(t')$. Consider B_0 to be the root of $\hat{\mathcal{T}}_0^s(t')$. The children blocks of B_0 in $\hat{\mathcal{T}}_0^s(t')$ are the descendent blocks at level c in $\hat{\mathcal{T}}_0(t')$. We can order these children blocks of B_0 in terms of their arrival times. Then, as the blocks in first $c-1$ levels of $\hat{\mathcal{T}}_0(t')$ are gift, the adversary didn't have to compute RandVDF.Eval for these blocks. Consider block B_1 to be the first block for which RandVDF.Eval was computed by the adversary. Therefore, the arrival time Q_1 of block B_1 is given by X_1 where X_1 is an exponential random variable in the static system $ss2$. On the other hand, the arrival time of the first child of B_1, call it $B_{1,1}$, is given by $Q_{1,1}=Q_1+X_{1,1}+\cdots+X_{1,c}$, where $X_{1,i}$ is the inter-arrival time between the $(i-1)^{th}$ and i^{th} descendent block of B_1. Note that, in the static system $ss2$, all the $X_{1,i}$'s are exponential with parameter λ_a, and they all are independent. Let the depth of the tree $\hat{\mathcal{T}}_0^s(t')$ be $D_0^s(t')$.

Each vertices at generation $k\geq 2$ in $\hat{\mathcal{T}}_0^s(t')$ can be labelled as a k tuple of positive integers $(i_1,...,i_k)$ with $i_j\geq c$ for $2\leq j\leq k$: the vertex $v=(i_1,...,i_k)\in\mathcal{I}_k$ is the (i_k-c+1)-th child of vertex $(i_1,...,i_{k-1})$ at level $k-1$. At $k=1$ generation, we have $i_1\geq 1$ as the adversary is gifted $c-1$ blocks on proposing the first block for which it computes only one RandVDF.Eval. Let $\mathcal{I}_k=\{(i_1,...,i_k): i_j\geq 1 \text{ for } i_j=1 \text{ and } i_j\geq c \text{ for } 2\leq j\leq k\}$, and set $\mathcal{I}=\cup_{k>0}\mathcal{I}_k$. For such v we also let $v^j=(i_1,...,i_j)$, $j=1,...,k$, denote the ancestor of v at level j, with $v^k=v$. For notation convenience, we set $v^0=0$ as the root of the tree.

Next, let $\{\mathcal{E}_v\}_{v\in\mathcal{I}}$ be an i.i.d. family of exponential random variables of parameter λ_a. For $v=(i_1,...,i_k)\in\mathcal{I}_k$, let $\mathcal{W}_v=\sum_{j\leq i_k}\mathcal{E}_{(i_1,...,i_{k-1},j)}$ and let $Q_v=\sum_{j\leq k}\mathcal{W}_{v^j}$. This creates a labelled tree, with the following interpretation: for $v=(i_1,...,i_j)$, the W_{v^j} are the waiting time for v^j to appear, measured from the appearance of v^{j-1}, and Q_v is the appearance time of v. Observe that starting from any $v\in\mathcal{I}_1$, we obtain a standard BRW. For any $v=(i_1,\cdots,i_k)\in\mathcal{I}_k$, we can write $Q_v=Q_v^1+Q_v^2$ where Q_v^1 is the appearance time for the ancestor of v at level 1 while $Q_v^2=Q_v-Q_v^1$.

Let $Q_k^*=\min_{v\in\mathcal{I}_k}Q_v$. Note that Q_k^* is the time of appearance of a block at level k and therefore we have

$$\{D_0(t')\geq ck\}=\{D_0^s(t')\geq k\}=\{Q_k^*\leq t'\}. \tag{2.16}$$

Fixing $i_1\in\mathcal{I}_1$, let $Q_{k,i_1}^{2*}=\min_{v\in\mathcal{I}_k \text{ s.t. } v^1=i_1}Q_v^2$. Observe that Q_{k,i_1}^{2*} is the minimum of a standard BRW with its root at the vertex i_1. Introduce, for $\theta_c<0$, the moment generating function

$$\begin{aligned}\Lambda_c(\theta_c) &= \log\sum_{\substack{v\in\mathcal{I}_2\\ v^1=i_1}}E(e^{\theta_c Q_v^2})=\log\sum_{j=c}^{\infty}E(e^{\sum_{i=1}^j\theta_c\mathcal{E}_i})\\ &= \log\sum_{j=c}^{\infty}(E(e^{\theta_c\mathcal{E}_1}))^j=\log\frac{E^c(e^{\theta_c\mathcal{E}_1})}{1-E(e^{\theta_c\mathcal{E}_1})}.\end{aligned}$$

Due to the exponential law of $\mathcal{E}_1$, $E(e^{\theta_c\mathcal{E}_1})=\frac{\lambda_a}{\lambda_a-\theta_c}$ and therefore $\Lambda_c(\theta_c)=\log(-\lambda_a^c/\theta_c(\lambda_a-\theta_c)^{c-1})$.

An important role is played by θ_c^*, which is the negative solution to the equation $\Lambda_c(\theta_c)=\theta_c\dot{\Lambda}_c(\theta_c)$ and let η_c satisfy that

$$\sup_{\theta_c<0}\left(\frac{\Lambda_c(\theta_c)}{\theta_c}\right)=\frac{\Lambda_c(\theta_c^*)}{\theta_c^*}=\frac{1}{\lambda_a\eta_c}.$$

Indeed, we have the following.

Proposition 1.

$$\lim_{k\to\infty}\frac{Q_k^*}{k}=\lim_{k\to\infty}\frac{Q_{k,i_1}^{2*}}{k}=\sup_{\theta_c<0}\left(\frac{\Lambda_c(\theta_c)}{\theta_c}\right)=\frac{1}{\lambda_a\eta_c},\quad a.s.$$

In fact, much more is known.

Proposition 2. *There exist explicit constants $c_1>c_2>0$ so that the sequence $Q_k^*-k/\lambda_a\eta_c-c_1\log k$ is tight, and*

$$\liminf_{k\to\infty}Q_k^*-k/\lambda_a\eta_c-c_2\log k=\infty, a.s.$$

Note that Propositions 1,2 and (2.16) imply in particular that $D_0(t') \le c\eta_c\lambda_a t'$ for all large t', a.s., and also that

$$\text{if } c\eta_c\lambda_a > \lambda_h \text{ then } D_i(t') > t' \text{ for all large } t', \text{ a.s..} \tag{2.17}$$

Let us define $\phi_c := c\eta_c$, then $\phi_c\lambda_a$ is the growth rate of private c-correlated NaS tree. With all these preparations, we can give a simple proof for Lemma 6.

Proof. Consider $m = \eta_c\lambda_a t' + x$. Note that by (2.16),

$$\begin{aligned} P(D_0^s(t') \ge m) &= P(Q_m^* \le t') \le \sum_{v\in\mathcal{I}_m} P(Q_v \le t') = \sum_{v\in\mathcal{I}_m} P(Q_v^1 + Q_v^2 \le t') \\ &= \sum_{v\in\mathcal{I}_m} \int_0^{t'} p_{Q_v^1}(u) P(Q_v^2 \le t'-u) du \\ &= \sum_{i_1\ge 1} \sum_{i_2\ge c,\ldots,i_m\ge c} \int_0^{t'} p_{Q_v^1}(u) P(Q_v^2 \le t'-u) du \end{aligned} \tag{2.18}$$

For $v = (i_1,\ldots,i_k)$, set $|v_{-1}| = i_2 + \cdots + i_k$. Then, we have that Q_v^2 has the same law as $\sum_{j=1}^{|v_{-1}|} \mathcal{E}_j$. Thus, by Chebycheff's inequality, for $v \in \mathcal{I}_m$,

$$P(Q_v^2 \le t'-u) \le E e^{\theta_c^* Q_v^2} e^{-\theta_c^*(t'-u)} = \left(\frac{\lambda_a}{\lambda_a - \theta_c^*}\right)^{|v_{-1}|} e^{-\theta_c^*(t'-u)}. \tag{2.19}$$

And

$$\begin{aligned} \sum_{i_2\ge c,\ldots,i_m\ge c} \left(\frac{\lambda_a}{\lambda_a-\theta_c^*}\right)^{|v_{-1}|} &= \sum_{i_2\ge c,\ldots,i_m\ge c} \left(\frac{\lambda_a}{\lambda_a-\theta_c^*}\right)^{\sum_{j=2}^m i_j} \\ &= \left(\sum_{i\ge c} \left(\frac{\lambda_a}{\lambda_a-\theta_c^*}\right)^i\right)^{m-1} = e^{(m-1)\Lambda_c(\theta_c^*)}. \end{aligned} \tag{2.20}$$

Combining (2.19), (2.20) and (2.18) yields

$$\begin{aligned} P(D_0^s(t') \ge m) &\le e^{-\theta_c^* t'} e^{(m-1)\Lambda_c(\theta_c^*)} \sum_{i_1\ge 1} \int_0^{t'} p_{Q_v^1}(u) e^{\theta_c^* u} du \\ &= e^{-\theta_c^* t'} e^{(m-1)\Lambda_c(\theta_c^*)} \sum_{i_1\ge 1} \int_0^{t'} \frac{\lambda_a^{i_1} u^{i_1-1} e^{-\lambda_a u}}{\Gamma(i_1)} e^{\theta_c^* u} du \\ &= e^{-\theta_c^* t'} e^{(m-1)\Lambda_c(\theta_c^*)} g(t'). \end{aligned} \tag{2.21}$$

where $g(t') = \sum_{i_1\ge 1} \int_0^{t'} \frac{\lambda_a^{i_1} u^{i_1-1} e^{-\lambda_a u}}{\Gamma(i_1)} e^{\theta_c^* u} du$. □

From proposition 1, we have

$$\phi_c = \frac{c\theta_c^*}{\lambda_a} \left(\frac{1}{\log\left(\frac{-\lambda_a^c}{\theta_c^*(\lambda_a-\theta_c^*)^{c-1}}\right)}\right), \tag{2.22}$$

where θ_c^* is the unique negative solution of

$$\Lambda_c(\theta_c) = \theta_c \dot{\Lambda}_c(\theta_c) \tag{2.23}$$

Note that $g(t')$ is an increasing function on t' and

$$\lim_{t'\to\infty} g(t') = \sum_{i_1\geq 1}\left(\frac{\lambda_a}{\lambda_a-\theta_c^\star}\right)^{i_1} = \frac{\lambda_a}{-\theta_c^\star} \tag{2.24}$$

Definitions and Preliminary Lemmas

In this section, we define some important events which will appear frequently in the analysis and provide some useful lemmas.

Let V_j be the event that the $j-$th honest block b_j is a loner, i.e.,

$$V_j = \{\tau_{j-1}^h < \tau_j^h - \Delta'\} \bigcap \{\tau_{j+1}^h > \tau_j^h + \Delta'\}$$

Let $\hat{F}_j = V_j \bigcap F_j$ be the event that b_j is a Nakamoto block. Then, we can define the following "potential" catch up event in $ss2$:

$$\hat{B}_{ik} = \{D_i(\tau_k^h + \Delta') \geq D_h(\tau_{k-1}^h) - D_h(\tau_i^h + \Delta')\}, \tag{2.25}$$

which is the event that the adversary launches a private attack starting from honest block b_i and catches up the fictitious honest chain right before honest block b_k is mined.

Lemma 12. *For each j,*

$$P(\hat{F}_j^c) = P(F_j^c \cup V_j^c) \leq P\left(\left(\bigcup_{(i,k):0\leq i<j<k} \hat{B}_{ik}\right) \cup V_j^c\right). \tag{2.26}$$

Proof.

$$\begin{aligned}
& P(V_j \cap E_{ij}) \\
= \; & P(V_j \cap \{D_i(t') < D_h(t'-\Delta') - D_h(\tau_i^h + \Delta') \text{ for all } t' > \tau_j^h + \Delta'\}) \\
= \; & P(V_j \cap \{D_i(t'+\Delta') < D_h(t') - D_h(\tau_i^h + \Delta') \text{ for all } t' > \tau_j^h\}\ ) \\
= \; & P(V_j \cap \{D_i(\tau_k^{h^-} + \Delta') < D_h(\tau_k^{h^-}) - D_h(\tau_i^h + \Delta') \text{ for all } k > j\}) \\
= \; & P(V_j \cap \{D_i(\tau_k^h + \Delta') < D_h(\tau_{k-1}^h) - D_h(\tau_i^h + \Delta') \text{ for all } k > j\}).
\end{aligned}$$

Since $\hat{F}_j = F_j \cap V_j = \bigcap_{0\leq i<j} E_{ij} \cap V_j$, by the definition of $\hat{B}_{ik}$ we have $P(\hat{F}_j) \geq P\left(\left(\bigcap_{(i,k):0\leq i<j<k} \hat{B}_{ik}^c\right) \cap V_j\right)$. Taking complement on both side, we can conclude the proof. □

Let $R_m = \tau_{m+1}^h - \tau_m^h$. Then, V_j and $\hat{B}_{ik}$ can be re-written as:

$$\begin{aligned}
V_j &= \{\Delta' < R_{j-1}\} \bigcap \{R_j > \Delta'\} \\
\hat{B}_{ik} &= \left\{D_i(\tau_i^h + \sum_{m=i}^{k-1} R_m + \Delta') \geq D_h(\tau_{k-1}^h) - D_h(\tau_i^h + \Delta')\right\}
\end{aligned} \tag{2.27}$$

Remark 1. *By time-warping, R_m is an IID exponential random variable with rate λ_h.*

Define X_d, $d > 0$, as the time it takes in the local clock of static system $ss2$ for D_h to reach depth d after reaching depth $d-1$. In other words, X_d is the difference between the times t_1 and t_2, where t_1 is the minimum time t' in the local clock of $ss2$ such that $D_h(t') = d$, and, t_2 is the minimum time t' in the local clock of $ss2$ such that $D_h(t') = d-1$.

Also, let $\delta_j^h = \tau_j^h - \tau_{j-1}^h$ and $\delta_j^a = \tau_j^a - \tau_{j-1}^a$ denote the inter-arrival time for honest and adversary arrival events in the local clock of static system $ss2$, respectively.

Proposition 3. *Let Y_d, $d \geq 1$, be i.i.d random variables, exponentially distributed with rate λ_h. Then, each random variable X_d can be expressed as $\Delta' + Y_d$.*

See Proposition $C.1$ in [69] for the proof.

Proposition 4. *For any constant a,*

$$P(\sum_{d=a}^{n+a} X_d > n(\Delta' + \frac{1}{\lambda_h})(1+\delta)) \leq e^{-n\Omega(\delta^2(1+\Delta'\lambda_h)^2)}$$

Proposition 4 is proved using chernoff bound and Proposition 3.

Proposition 5. *Probability that there are less than*

$$n\frac{\lambda_a(1-\delta)}{\lambda_h}$$

adversarial arrival events for which RandVDF.Eval *has been computed in the interval τ_0^h to τ_{n+1}^h is upper bounded by*

$$e^{-n\Omega(\delta^2 \frac{\lambda_a}{\lambda_h})}$$

Proposition 5 is proven using the Poisson tail bounds.

Proposition 6. *For $n > \frac{c-1}{\phi_c - 1}$, define B_n as the event that there are at least n adversarial block arrivals for each of which adversary computed* RandVDF.Eval *while D_h grows from depth 0 to $n+c-1$:*

$$B_n = \{ \sum_{i=1}^{n+c-1} X_i \geq \sum_{i=0}^{n} \delta_i^a \}$$

If

$$\phi_c \lambda_a < \frac{\lambda_h}{1+\lambda_h \Delta'},$$

then,

$$P(B_n) \leq e^{-A_1 n} e^{-A_2}$$

,

$$A_1 = -w\Delta' + \ln\left(\frac{\lambda_a + w}{\lambda_a}\right) + \ln\left(\frac{\lambda_h - w}{\lambda_h}\right)$$
$$A_2 = -(c-1)w\Delta' + (c-1)\ln\left(\frac{\lambda_h - w}{\lambda_h}\right)$$

such that $A_1 + \frac{A_2}{n} > 0$ and,

$$w = \frac{\lambda_h - \lambda_a}{2} + \frac{2n+c-1}{2(n+c-1)\Delta'} - \frac{\sqrt{[(n+c-1)\Delta'(\lambda_a - \lambda_h)]^2 + (2n+c-1)^2 + 2(n+c-1)\Delta'[(c-1)(\lambda_a+\lambda_h) + 2(n+c-1)\Delta'\lambda_a\lambda_h]}}{2(n+c-1)\Delta'}$$

Proof. Using Chebychev inequality and proposition 3, for any $t > 0$, we have

$$\begin{aligned} P(B_n) &\leq E\left[\prod_{j=0}^{n} e^{-w\delta_i^a}\right] E\left[\prod_{j=1}^{n+c-1} e^{wX_i}\right] \\ &\leq \left[\frac{\lambda_a}{\lambda_a + w}\right]^n \left[\frac{e^{w\Delta'}\lambda_h}{\lambda_h - w}\right]^{n+c-1} \\ &= e^{-n\left[-\left(\frac{n+c-1}{n}\right)w\Delta' + \left(\frac{n+c-1}{n}\right)\ln\left(\frac{\lambda_h - w}{\lambda_h}\right) + \ln\left(\frac{\lambda_a + w}{\lambda_a}\right)\right]} \end{aligned}$$

Optimizing over w, we have

$$\frac{d}{dw}\left[-\left(\frac{n+c-1}{n}\right)w\Delta'+\left(\frac{n+c-1}{n}\right)\ln\left(\frac{\lambda_h-w}{\lambda_h}\right)+\ln\left(\frac{\lambda_a+w}{\lambda_a}\right)\right]=0$$

$$(n+c-1)\Delta' w^2+[(n+c-1)\Delta'(\lambda_a-\lambda_h)-(2n+c-1)]w$$
$$+[n\lambda_h-(n+c-1)\lambda_a-(n+c-1)\Delta'\lambda_a\lambda_h]=0$$

$$w=\frac{\lambda_h-\lambda_a}{2}+\frac{2n+c-1}{2(n+c-1)\Delta'}-$$
$$\frac{\sqrt{[(n+c-1)\Delta'(\lambda_a-\lambda_h)]^2+(2n+c-1)^2+2(n+c-1)\Delta'[(c-1)(\lambda_a+\lambda_h)+2(n+c-1)\Delta'\lambda_a\lambda_h]}}{2(n+c-1)\Delta'}$$

Note that for $n>\frac{c-1}{\phi_c-1}$, we have $\lambda_a\left(1+\frac{c-1}{n}\right)<\phi_c\lambda_h<\frac{\lambda_h}{1+\Delta'\lambda_h}$. That implies $w>0$.

Also, using $n>\frac{c-1}{\phi_c-1}$, we have

$$-\left(\frac{n+c-1}{n}\right)w\Delta'+\left(\frac{n+c-1}{n}\right)\ln\left(\frac{\lambda_h-w}{\lambda_h}\right)+\ln\left(\frac{\lambda_a+w}{\lambda_a}\right)=A_1+\frac{A_2}{n}>0$$

□

Lemma 13. *For $k-i>\frac{\lambda_h(c-1)}{\lambda_a(\phi_c-1)}$, there exists a constant $\gamma>0$ such that*

$$P(\hat{B}_{ik})\leq e^{-\gamma(k-i)} \tag{2.28}$$

Proof. Let $N(\tau_i^h,\tau_k^h+\Delta')$ be the number of adversarial arrivals for which RandVDF.Eval in was computed in $ss2$ in the interval $[\tau_i^h,\tau_k^h+\Delta']$. Define

$$\hat{C}_{ik}=\text{ event that } N(\tau_i^h,\tau_k^h+\Delta')+(c-1)\geq D_h(\tau_{k-1}^h)-D_h(\tau_i^h+\Delta')$$

Observe that $D_i(\tau_i^h,\tau_k^h+\Delta')\leq N(\tau_i^h,\tau_k^h+\Delta')+(c-1)$, where $c-1$ is due to the fact that blocks in first $c-1$ levels are gifted to the adversary on proposing the first block in the adversarial tree. Note that RandVDF.Eval was not computed by the adversary for these $c-1$ blocks. Then, we have

$$\hat{B}_{ik}\subseteq\hat{C}_{ik}.$$

$$\begin{aligned}
P(\hat{B}_{ik}) &\leq P\left(N(\tau_i^h,\tau_k^h+\Delta')<(1-\delta)(k-i)\frac{\lambda_a}{\lambda_h}\right)\\
&+ P\left(\hat{C}_{ik}\,|\,N(\tau_i^h,\tau_k^h+\Delta')\geq(1-\delta)(k-i)\frac{\lambda_a}{\lambda_h}\right)\\
&\overset{(a)}{\leq} e^{-\Omega((k-i)\delta^2\lambda_a/\lambda_h)}+P\left(\hat{C}_{ik}\,|\,N_a(\tau_i^h,\tau_k^h+\Delta')\geq(1-\delta)(k-i)\frac{\lambda_a}{\lambda_h}\right)\\
&\overset{(b)}{\leq} e^{-\Omega((k-i)\delta^2\lambda_a/\lambda_h)}+\\
&\qquad\sum_{x=(1-\delta)(k-i)\frac{\lambda_a}{\lambda_h}}^{\infty} P\big(D_h(\tau_{k-1}^h)-D_h(\tau_i^h+\Delta')\leq x+c-1\,|\,N_a(\tau_i^h,\tau_k^h+\Delta')=x\big)\\
&\overset{(c)}{=} e^{-\Omega((k-i)\delta^2\lambda_a/\lambda_h)}+\sum_{x=(1-\delta)(k-i)\frac{\lambda_a}{\lambda_h}}^{\infty} e^{-A_1x}e^{-A_2}\\
&\overset{(d)}{=} e^{-\Omega((k-i)\delta^2\lambda_a/\lambda_h)}+e^{-A_2}\frac{1}{1-e^{-A_3}}e^{-A_3(k-i)}
\end{aligned}$$

where (a) is due to proposition 5 which says that there are more than $(1-\delta)(k-i)\lambda_a/\lambda_h$ adversarial

arrival events in the time period $[\tau_i^h, \tau_k^h + \Delta']$ except with probability $e^{-\Omega((k-i)\delta^2\lambda_a/\lambda_h)}$, (b) is by union bound, (c) is by proposition 6 for $k-i > \frac{\lambda_h(c-1)}{\lambda_a(\phi_c-1)}$, (d) is due to $A_3 = \frac{A_1(1-\delta)\lambda_a}{\lambda_h}$.

Hence,

$$P(\hat{B}_{ik}) < C_1 e^{-C_2(k-i)} \tag{2.29}$$

for appropriately chosen constants $C_1, C_2, > 0$ as functions of the fixed δ. Finally, since $P(\hat{B}_{ik})$ decreases as $k-i$ grows and is smaller than 1 for sufficiently large $k-i$, we obtain the desired inequality for a sufficiently small $\gamma \leq C_3$.

□

2.8.5 Proof of Lemma 7

For notational convenience, we will continue to use τ_i^h and τ_i^a as the arrival time of the $i-th$ honest and adversarial blocks in the static system $ss2$, respectively. In this proof, let $r_h := \frac{\lambda_h}{1+\lambda_h\Delta'}$. The random processes of interest start from time 0. To look at the system in stationarity, let us extend them to $-\infty < t' < \infty$. More specifically, define $\tau_{-1}^h, \tau_{-2}^h, \ldots$ such that together with $\tau_0^h, \tau_1^h, \ldots$, we have a double-sided infinite random process. Also, for each $i < 0$, we define an independent copy of a random adversary tree $\hat{\mathcal{T}}_i$ with the same distribution as $\hat{\mathcal{T}}_0$. And we extend the definition of $\hat{\mathcal{T}}_h(t')$ and $D_h(t')$ to $t' < 0$: the last honest block mined at $\tau_{-1}^h < 0$ and all honest blocks mined within $(\tau_{-1}^h - \Delta', \tau_{-1}^h)$ appear in $\hat{\mathcal{T}}_h(t')$ at their respective mining times to form the level -1, and the process repeats for level less than -1; let $D_h(t')$ be the level of the last honest arrival before t' in $\hat{\mathcal{T}}_h(t')$, i.e., $D_h(t') = \ell$ if $\tau_i^h \leq t' < \tau_{i+1}^h$ and the i-th honest block appears at level ℓ of $\hat{\mathcal{T}}_h(t)$.

These extensions allow us to extend the definition of E_{ij} to all i, j, $-\infty < i < j < \infty$, and define E_j and $\hat{E}_j$ to be:

$$E_j = \bigcap_{i<j} E_{ij}$$

and

$$\hat{E}_j = E_j \cap V_j.$$

Note that $\hat{E}_j \subset \hat{F}_j$, so to prove that $\hat{F}_j$ has a probability bounded away from 0 for all j, all we need is to prove that $\hat{E}_j$ has a non-zero probability.

Recall that we have defined the events V_j and $\hat{B}_{ik}$ in section 2.8.4 of the appendix as:

$$V_j = \{\Delta' < R_{j-1}\} \bigcap \{R_j > \Delta'\}$$
$$\hat{B}_{ik} = \left\{ D_i(\tau_i^h + \sum_{m=i}^{k-1} R_m + \Delta') \geq D_h(\tau_{k-1}^h) - D_h(\tau_i^h + \Delta') \right\}$$

where R_m are i.i.d exponential random variable with mean $\frac{1}{\lambda_h}$.

Following the idea in Lemma 12 and using Lemma 14 and 15, we have

$$P(E_j \cap V_j) = P\left(\bigcap_{i<j} E_{ij} \cap V_j \right) = P\left(\left(\bigcap_{i<j<k} \hat{B}_{ik}^c \right) \cap U_j \right).$$

where $E_j = \bigcap_{i<j<k} \hat{B}_{ik}^c$ and $\hat{E}_j = E_j \cap U_j$. So, we just need to prove that $\hat{E}_j$ has a non-zero probability. Observe that, due to constant adversarial and honest mining rate and the growth rate of the adversarial tree being independent of level of its root in the static system $ss2$, $\hat{E}_j$ has a time-invariant dependence on $\{\mathcal{Z}_i\}$, which means that $p = P(\hat{E}_j)$ does not depend on j. Then we can just focus on $P(\hat{E}_0)$. This

is the last step to prove.

$$\begin{aligned} P(\hat{E}_0) &= P(E_0|U_0)P(U_0) \\ &= P(E_0|U_0)P(R_0>\Delta')P(R_{-1}>\Delta') \\ &= e^{-2\lambda_h\Delta'}P(E_0|U_0). \end{aligned}$$

where we used Remark 1 in the last step. It remains to show that $P(E_0|U_0)>0$. We have

$$\begin{aligned} E_0 &= \text{event that } D_i(\sum_{m=i}^{k-1} R_m+\Delta'+\tau_i^h) < D_h(\tau_{k-1}^h)-D_h(\tau_i^h+\Delta') \\ &\quad \text{for all } k>0 \text{ and } i<0, \end{aligned}$$

then

$$(E_0)^c = \bigcup_{k>0,i<0} \hat{B}_{ik}. \tag{2.30}$$

Let us fix a particular $n>2\lambda_h\Delta'>0$, and define:

$$\begin{aligned} G_n &= \text{event that} D_m(3n/\lambda_h+\zeta_m^h)=0 \\ &\quad \text{for } m=-n,-n+1,...,-1,0,+1,...,n-1,n \end{aligned}$$

Then

$$\begin{aligned} P(E_0|U_0) &\geq P(E_0|U_0,G_n)P(G_n|U_0) \\ &= \Big(1-P(\cup_{k>0,i<0}\hat{B}_{ik}|U_0,G_n)\Big)P(G_n|U_0) \\ &\geq \left(1-\sum_{k>0,i<0} P(\hat{B}_{ik}|U_0,G_n)\right)P(G_n|U_0) \\ &\geq (1-a_n-b_n)P(G_n|U_0) \end{aligned} \tag{2.31}$$

where

$$a_n := \sum_{(i,k):-n\leq i<0<k\leq n} P(\hat{B}_{ik}|U_0,G_n) \tag{2.32}$$

$$b_n := \sum_{(i,k):i<-n \text{ or } k>n} P(\hat{B}_{ik}|U_0,G_n). \tag{2.33}$$

Consider two cases:

Case 1: $-n \le i < 0 < k \le n$:

$$\begin{aligned}
P(\hat{B}_{ik}|U_0,G_n) &= P(\hat{B}_{ik}|U_0,G_n,\sum_{m=i}^{k-1} R_m + \Delta' \le 3n/\lambda_h) \\
&\quad + P(\sum_{m=i}^{k-1} R_m + \Delta' > 3n/\lambda_h|U_0,G_n) \\
&\le P(\sum_{m=i}^{k-1} R_m + \Delta' > 3n/\lambda_h|U_0,G_n) \\
&\le P(\sum_{m=i}^{k-1} R_m > 5n/(2\lambda_h)|U_0) \\
&\le P(\sum_{m=i}^{k-1} R_m > 5n/(2\lambda_h))/P(U_0) \\
&\le A_5 e^{-\gamma_1 n}
\end{aligned}$$

for some positive constants A_5, γ_1 independent of n,k,i. The last inequality follows from the fact that R_i's are iid exponential random variables of mean $1/\lambda_h$. Summing these terms, we have:

$$\begin{aligned}
a_n &= \sum_{(i,k):-n\le i<0<k\le n} P(B_{ik}|U_0,G_n) \\
&\le \sum_{(i,k):-n\le i<0<k\le n} A_5 e^{-\alpha_1 n} := \bar{a}_n,
\end{aligned}$$

which is bounded and moreover $\bar{a}_n \to 0$ as $n \to \infty$.

Case 2: $k > n$ or $i < -n$:

For $0 < \varepsilon < 1$, let us define event W_{ik}^{ε} to be:

$$W_{ik}^{\varepsilon} = \text{event that } D_h(\zeta_{k-1}^h) - D_h(\zeta_i^h + \Delta') \ge (1-\varepsilon)\tfrac{r_h}{\lambda_h}(k-i-1). \tag{2.34}$$

Then we have

$$P(\hat{B}_{ik}|U_0,G_n) \le P(\hat{B}_{ik}|U_0,G_n,W_{ik}^{\varepsilon}) + P({W_{ik}^{\varepsilon}}^c|U_0,G_n).$$

We first bound $P({W_{ik}^{\varepsilon}}^c|U_0,G_n)$:

$$\begin{aligned}
P({W_{ik}^{\varepsilon}}^c|U_0,G_n) &\le P({W_{ik}^{\varepsilon}}^c|\zeta_{k-1}^h - \zeta_i^h - \Delta' > \frac{k-i-1}{(1+\varepsilon)\lambda_h}) \\
&\quad + P(\zeta_{k-1}^h - \zeta_i^h - \Delta' \le \frac{k-i-1}{(1+\varepsilon)\lambda_h}) \\
&\le P({W_{ik}^{\varepsilon}}^c|\zeta_{k-1}^h - \zeta_i^h - \Delta' > \frac{k-i-1}{(1+\varepsilon)\lambda_h}) + e^{-\Omega(\varepsilon^2(k-i-1))} \\
&\le e^{-\Omega(\varepsilon^4(k-i-1))} + e^{-\Omega(\varepsilon^2(k-i-1))} \\
&\le A_6 e^{-\gamma_2(k-i-1)}
\end{aligned} \tag{2.35}$$

for some positive constants A_6, γ_2 independent of n,k,i, where the second inequality follows from the Erlang tail bound (as $\zeta_{k-1}^h - \zeta_i^h$ is sum of IID exponentials due to time-warping) and the third inequality follows from Proposition 4.

Meanwhile, we have

$$
\begin{aligned}
& P(\hat{B}_{ik}|U_0,G_n,W_{ik}^{\varepsilon}) \\
\leq\ & P(D_i(\sum_{m=i}^{k-1}R_m+\Delta'+\zeta_i^h)\geq(1-\varepsilon)\frac{r_h}{\lambda_h}(k-i-1)|U_0,G_n,W_{ik}^{\varepsilon}) \\
\leq\ & P(D_i(\sum_{m=i}^{k-1}R_m+\Delta'+\zeta_i^h)\geq(1-\varepsilon)\frac{r_h}{\lambda_h}(k-i-1) \\
& \quad |U_0,G_n,W_{ik}^{\varepsilon},\sum_{m=i}^{k-1}R_m+\Delta'\leq(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}) \\
& \quad +P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}|U_0,G_n,W_{ik}^{\varepsilon}) \\
\overset{(a)}{\leq}\ & P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}|U_0,G_n,W_{ik}^{\varepsilon}) \\
& \quad +e^{-\theta_c^*(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}+\left(\frac{(1-\varepsilon)}{c}\frac{r_h}{\lambda_h}(k-i-1)-1\right)\Lambda_c(\theta_c^*)}g\left((k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}\right) \\
\overset{(b)}{=}\ & P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}|U_0,G_n,W_{ik}^{\varepsilon}) \\
& \quad +e^{-\theta_c^*\frac{k-i-1}{\lambda_h}\left[\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}-(1-\varepsilon)\frac{r_h}{\phi_c\lambda_a}\right]}e^{-\Lambda_c(\theta_c^*)}g\left((k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}\right)
\end{aligned}
$$

where (a) follows from Lemma 6, (b) follows from $\frac{\Lambda_c(\theta_c^*)}{\theta_c^*}=\frac{1}{\lambda_a\eta_c}=\frac{c}{\phi_c\lambda_a}$. The first term can be bounded as:

$$
\begin{aligned}
& P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}|U_0,G_n,W_{ik}^{\varepsilon}) \\
=\ & P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}|U_0,W_{ik}^{\varepsilon}) \\
\leq\ & P(\sum_{m=i}^{k-1}R_m+\Delta'>(k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h})/P(U_0,W_{ik}^{\varepsilon}) \\
\leq\ & A_7e^{-\gamma_3(k-i-1)}
\end{aligned}
$$

for some positive constants A_7,γ_3 independent of n,k,i. The last inequality follows from the fact that $(r_h+\phi_c\lambda_a)/(2\phi_c\lambda_a)>1$ and the R_i's have mean $1/\lambda_h$, while $P(U_0,W_{ik}^{\varepsilon})$ is a event with high probability as we showed in (2.35). Then we have

$$
\begin{aligned}
& P(\hat{B}_{ik}|U_0,G_n) \\
\leq\ & A_6e^{-\alpha_2(k-i-1)} \\
+\ & e^{-\theta_c^*(k-i-1)\frac{r_h(1-\varepsilon)}{\lambda_h\phi_c\lambda_a}\left[\frac{r_h+\phi_c\lambda_a}{2(1-\epsilon)r_h}-1\right]}e^{-\Lambda_c(\theta_c^*)}g((k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}) \\
+\ & A_7e^{-\gamma_3(k-i-1)}. \qquad (2.36)
\end{aligned}
$$

Summing these terms, we have:

$$\begin{aligned} b_n &= \sum_{(i,k):i<-n \text{ or } k>n} P(\hat{B}_{ik}|U_0,G_n) \\ &\leq \sum_{(i,k):i<-n \text{ or } k>n} [A_6 e^{-\alpha_2(k-i-1)} \\ &\quad + e^{-\theta_c^*(k-i-1)\frac{r_h(1-\varepsilon)}{\lambda_h \phi_c \lambda_a}\left[\frac{r_h+\phi_c\lambda_a}{2(1-\epsilon)r_h}-1\right]} e^{-\Lambda_c(\theta_c^*)} g\left((k-i-1)\frac{r_h+\phi_c\lambda_a}{2\phi_c\lambda_a}\frac{1}{\lambda_h}\right) \\ &\quad + A_7 e^{-\gamma_3(k-i-1)}] \\ &:= \bar{b}_n \end{aligned}$$

Here, from (2.24), $g(.)\to\frac{\lambda_a}{-\theta_c^*}$ as $n\to\infty$. Therefore, $\bar{b}_n$ is bounded and moreover $\bar{b}_n\to 0$ as $n\to\infty$ when we set ε to be small enough such that $\frac{r_h+\phi_c\lambda_a}{2(1-\varepsilon)r_h}<1$.

Substituting these bounds in (2.31) we finally get:

$$P(E_0|U_0) > [1-(\bar{a}_n+\bar{b}_n)]P(G_n|U_0) \tag{2.37}$$

By setting n sufficiently large such that $\bar{a}_n$ and $\bar{b}_n$ are sufficiently small, we conclude that $P(\hat{E}_0)>0$.

2.8.6 Proof of Lemma 8

We divide the proof in to two steps. In the first step, we prove for $\varepsilon=1/2$. Recall that we have defined event $\hat{B}_{ik}$ as:

$$\hat{B}_{ik} = \text{event that } D_i(\textstyle\sum_{m=i}^{k-1} R_m+\Delta'+\zeta_i^h) \geq D_h(\zeta_{k-1}^h) - D_h(\zeta_i^h+\Delta').$$

And by Lemma 14, 15, 12, we have

$$\hat{F}_j^c = F_j^c \cup V_j^c = \left(\bigcup_{(i,k):i<j<k} \hat{B}_{ik}\right) \cup V_j^c. \tag{2.38}$$

For $t' > \max\left\{\left(\frac{2\lambda_h}{1-\eta}\right)^2\left(\frac{c-1}{\phi_c-1}\right)^2, \left[(c-1)\left(\Delta'+\frac{1}{\lambda_{\min}}\right)\right]^2\right\}$, we have $\frac{\sqrt{t'}}{2\lambda_h} > \frac{\lambda_h}{\lambda_a}\left(\frac{c-1}{\phi_c-1}\right)$ and $\sqrt{t'} > (c-1)\left(\Delta'+\frac{1}{\lambda_{\min}}\right)$.

Divide $[s',s'+t']$ into $\sqrt{t'}$ sub-intervals of length $\sqrt{t'}$, so that the r th sub-interval is:

$$\mathcal{J}_r := [s'+(r-1)\sqrt{t'}, s'+r\sqrt{t'}].$$

Now look at the first, fourth, seventh, etc sub-intervals, i.e. all the $r=1 \mod 3$ sub-intervals. Introduce the event that in the ℓ-th $1 \mod 3$th sub-interval, an adversary tree that is rooted at a honest block arriving in that sub-interval or in the previous ($0 \mod 3$) sub-interval catches up with a honest block in that sub-interval or in the next ($2 \mod 3$) sub-interval. Formally,

$$C_\ell = \bigcap_{j:\zeta_j^h \in \mathcal{J}_{3\ell+1}} U_j^c \cup \left(\bigcup_{(i,k):\zeta_j^h-\sqrt{t'}<\zeta_i^h<\zeta_j^h, \zeta_j^h<\zeta_k^h+\Delta'<\zeta_j^h+\sqrt{t'}} \hat{B}_{ik}\right).$$

We have

$$P(C_\ell) \leq P(\text{no arrival in } \mathcal{J}_{3\ell+1}) + 1 - p < 1 \tag{2.39}$$

for large enough t', where p is a uniform lower bound such that $P(\hat{F}_j)\geq p$ for all j. Also, we define the

following event:

$$\hat{C}_\ell = \text{ event that the honest fictitious tree grows by } c-1 \text{ levels in sub-interval } \mathcal{J}_{3\ell+2}$$

Observe that because of `randSource` being updated at each epoch beginning, for distinct ℓ, the events $C_\ell \bigcap \hat{C}_\ell$ are independent. Using Chernoff bounds, for $\sqrt{t'} > (c-1)\left(\Delta' + \frac{1}{\lambda_{\min}}\right)$, we have $P(\hat{C}_\ell) \geq 1 - e^{-c_2\sqrt{t'}}$.

Introduce the atypical events:

$$\begin{aligned} B &= \bigcup_{(i,k):\zeta_i^h \in [s',s'+t'] \text{ or } \zeta_k^h+\Delta' \in [s',s'+t'], i<k, \zeta_k^h+\Delta'-\zeta_i^h > \sqrt{t'}} \hat{B}_{ik}, \\ \text{and} \\ \tilde{B} &= \bigcup_{(i,k):\zeta_i^h < s', s'+t' < \zeta_k^h+\Delta'} \hat{B}_{ik}. \end{aligned}$$

The events B and $\tilde{B}$ are superset of the events that an adversary tree catches up with an honest block far ahead. Then we have

$$\begin{aligned} P(B^{static}_{s',s'+t'}) &\leq P\Big(\bigcap_{j:\zeta_j^h \in [s',s'+t']} U_j^c\Big) + P(B) + P(\tilde{B}) + P\Big(\bigcap_{\ell=0}^{\sqrt{t'}/3} C_\ell\Big) \\ &\leq P\Big(\bigcap_{j:\zeta_j^h \in [s',s'+t']} U_j^c\Big) + P(B) + P(\tilde{B}) + P\Big(\bigcup_{\ell=0}^{\sqrt{t'}/3} \hat{C}_\ell^c\Big) + P\Big(\bigcap_{\ell=0}^{\sqrt{t'}/3} C_\ell \cap \hat{C}_\ell\Big) \\ &\leq P\Big(\bigcap_{j:\zeta_j^h \in [s',s'+t']} U_j^c\Big) + P(B) + P(\tilde{B}) + \sum_{\ell=0}^{\sqrt{t'}/3} P(\hat{C}_\ell^c) + (P(C_\ell \cap \hat{C}_\ell))^{\sqrt{t'}/3} \\ &\leq e^{-c_1 t'} + P(B) + P(\tilde{B}) + e^{-c_2\sqrt{t'}} + (P(C_\ell))^{\frac{\sqrt{t'}}{3}} \end{aligned} \tag{2.40}$$

for some positive constants c_1, c_2 when t' is large. Next we will bound the atypical events B and $\tilde{B}$. Consider the following events

$$\begin{aligned} D_1 &= \{\#\{i : \zeta_i^h \in (s'-\sqrt{t'}-\Delta', s'+t'+\sqrt{t'}+\Delta')\} > 2\lambda_h t'\} \\ D_2 &= \{\exists i,k : \zeta_i^h \in (s', s'+t'), (k-i) < \frac{\sqrt{t'}}{2\lambda_h}, \zeta_k^h - \zeta_i^h + \Delta' > \sqrt{t'}\} \\ D_3 &= \{\exists i,k : \zeta_k^h + \Delta \in (s', s'+t'), (k-i) < \frac{\sqrt{t'}}{2\lambda_h}, \zeta_k^h - \zeta_i^h + \Delta' > \sqrt{t}\} \end{aligned}$$

In words, D_1 is the event of atypically many honest arrivals in $(s' - \sqrt{t'} - \Delta', s' + t' + \sqrt{t'} + \Delta')$ while D_2 and D_3 are the events that there exists an interval of length $\sqrt{t'}$ with at least one endpoint inside $(s', s' + t')$ with atypically small number of arrivals. Since, by time-warping, the number of honest arrivals in $(s', s' + t')$ (in the local clock of the static system) is Poisson with parameter $\lambda_h t'$, we have from the memoryless property of the Poisson process that $P(D_1) \leq e^{-c_0 t'}$ for some constant $c_0 = c_0(\lambda_a, \lambda_h) > 0$ when t' is large. On the other hand, using the memoryless property and a union bound, and decreasing c_0 if needed, we have that $P(D_2) \leq e^{-c_0\sqrt{t'}}$. Similarly, using time reversal,

$P(D_3)\leq e^{-c_0\sqrt{t'}}$. Therefore, again using the memoryless property of the Poisson process,

$$\begin{aligned} P(B) &\leq P(D_1\cup D_2\cup D_3)+P(B\cap D_1^c\cap D_2^c\cap D_3^c) \\ &\leq e^{-c_0t'}+2e^{-c_0\sqrt{t'}}+\sum_{i=1}^{2\lambda_h t'}\sum_{k:k-i>\sqrt{t'}/2\lambda_h} P(\hat{B}_{ik}) \qquad (2.41) \\ &\leq e^{-c_3\sqrt{t'}}, \qquad (2.42) \end{aligned}$$

for large t', where $c_3>0$ are constants that may depend on λ_a,λ_h and the last inequality is due to (2.28). We next claim that there exists a constant $\alpha>0$ such that, for all t' large,

$$P(\tilde{B})\leq e^{-c_6t'}. \qquad (2.43)$$

Consider the following event

$$D_4=\{\exists i,k:\zeta_i^h<s',s'+t'<\zeta_k^h+\Delta',(k-i)<\frac{t'}{2\lambda_h},\zeta_k^h-\zeta_i^h+\Delta'>t'\}.$$

Using Poisson tail bounds, we can show that $P(D_4)\leq e^{-c_4t'}$. Now, we have

$$\begin{aligned} & P(\tilde{B})\leq P(D_4)+P(\tilde{B}\cap D_4^c) \\ \leq\ & e^{-c_4t'}+\sum_{i,k:k-i>t'/2\lambda_h}\int_0^{s'} P(\zeta_i^h\in d\theta)P(\hat{B}_{ik},\zeta_k^h-\zeta_i^h+\Delta'>s'+t'-\theta) \\ \leq\ & e^{-c_4t'}+\sum_i\int_0^{s'} P(\zeta_i^h\in d\theta)\sum_{k:k-i>t'/2\lambda_h} P(\hat{B}_{ik})^{1/2}P(\zeta_k^h-\zeta_i^h+\Delta'>s'+t'-\theta)^{1/2}. \end{aligned} \qquad (2.44)$$

The tails of the Poisson distribution yield the existence of constants $c',c''>0$ so that

$$\begin{aligned} & P(\zeta_k^h-\zeta_i^h+\Delta'>s'+t'-\theta) \qquad (2.45) \\ \leq\ & \begin{cases} 1, & (k-i)>c'(s'+t'-\theta-\Delta') \\ e^{-c''(s'+t'-\theta-\Delta')}, & (k-i)\leq c'(s'+t'-\theta-\Delta'). \end{cases} \qquad (2.46) \end{aligned}$$

(2.28) and (2.45) yield that, for large enough t', there exists a constant $c_5>0$ so that

$$\sum_{k:k-i>t'/2\lambda_h} P(\hat{B}_{i,k})^{1/2}P(\zeta_k^h-\zeta_i^h>s'+t'-\theta-\Delta')^{1/2}\leq e^{-2c_5(s'+t'-\theta-\Delta')}. \qquad (2.47)$$

Substituting this bound in (2.44) and using that $\sum_i P(\zeta_i^h\in d\theta)=d\theta$ gives

$$\begin{aligned} P(\tilde{B}) &\leq e^{-c_4t'}+\sum_i\int_0^{s'} P(\zeta_i^h\in d\theta)e^{-2c_5(s'+t'-\theta-\Delta')} \\ &\leq e^{-c_4t'}+\int_0^{s'} e^{-2c_5(s'+t'-\theta-\Delta')}d\theta\leq e^{-c_4t'}+\frac{1}{2c_5}e^{-2c_5(t'-\Delta')} \\ &\leq e^{-c_6t'}, \qquad (2.48) \end{aligned}$$

for t' large and $c_6=\min(c_4,c_5)$, proving (2.43).

Combining (2.42), (2.48) and (2.40) concludes the proof of step 1.

In step two, we prove for any $\varepsilon>0$ by recursively applying the bootstrapping procedure in step 1.

Assume the following statement is true: for any $\theta \geq m$ there exist constants $\bar{b}_\theta, \bar{A}_\theta$ so that for all $s', t' \geq 0$,

$$\tilde{q}[s', s'+t'] \leq \bar{A}_\theta \exp(-\bar{b}_\theta t'^{1/\theta}). \tag{2.49}$$

By step 1, it holds for $m=2$. Also, for specific values of m that we will consider, we will have $t'^{\frac{m}{2m-1}} > \sqrt{t'}$.

Divide $[s', s'+t']$ into $t'^{\frac{m-1}{2m-1}}$ sub-intervals of length $t'^{\frac{m}{2m-1}}$, so that the r th sub-interval is:

$$\mathcal{J}_r := [s'+(r-1)t'^{\frac{m}{2m-1}}, s'+rt'^{\frac{m}{2m-1}}].$$

Now look at the first, fourth, seventh, etc sub-intervals, i.e. all the $r = 1 \mod 3$ sub-intervals. Introduce the event that in the ℓ-th $1 \mod 3$th sub-interval, an adversary tree that is rooted at a honest block arriving in that sub-interval or in the previous ($0 \mod 3$) sub-interval catches up with a honest block in that sub-interval or in the next ($2 \mod 3$) sub-interval. Formally,

$$C_\ell = \bigcap_{j:\zeta_j^h \in \mathcal{J}_{3\ell+1}} U_j^c \cup \left(\bigcup_{(i,k):\zeta_j^h - t'^{\frac{m}{2m-1}} < \zeta_i^h < \zeta_j^h, \zeta_j^h < \zeta_k^h + \Delta' < \zeta_j^h + t'^{\frac{m}{2m-1}}} \hat{B}_{ik} \right).$$

By (2.49), we have

$$P(C_\ell) \leq A_m \exp(-\bar{a}_m t'^{\frac{1}{2m-1}}). \tag{2.50}$$

Also, we define the following event:

$$\hat{C}_\ell = \text{ event that the honest fictitious tree grows by } c-1 \text{ levels in sub-interval } \mathcal{J}_{3\ell+2}$$

Note that for distinct ℓ, the events $C_\ell \bigcap \hat{C}_\ell$ are independent. Also, from Lemma 10, assuming $t'^{\frac{m}{2m-1}} > \sqrt{t'} > (c-1)\left(\Delta' + \frac{1}{\lambda_{\min}}\right)$, we have $P(\hat{C}_\ell) \geq 1 - e^{-c_2 t'^{\frac{m}{2m-1}}}$ for some positive constant c_2.

Introduce the atypical events:

$$B = \bigcup_{(i,k):\zeta_i^h \in [s', s'+t'] \text{ or } \zeta_k^h + \Delta' \in [s', s'+t'], i<k, \zeta_k^h + \Delta' - \zeta_i^h > t'^{\frac{m}{2m-1}}} \hat{B}_{ik},$$

and

$$\tilde{B} = \bigcup_{(i,k):\zeta_i^h < s', s'+t' < \zeta_k^h + \Delta'} \hat{B}_{ik}.$$

The events B and $\tilde{B}$ are the events that an adversary tree catches up with an honest block far ahead. Following the calculations in step 1, we have

$$P(B) \leq e^{-c_3 t'^{\frac{m}{2m-1}}} \tag{2.51}$$

$$P(\tilde{B}) \leq e^{-c_6 t'}, \tag{2.52}$$

for large t', where c_1 and c_5 are some positive constant.

Then we have

$$\begin{aligned}
\tilde{q}[s',s'+t'] &\leq P(\bigcap_{j:\zeta_j^h\in[s',s'+t']} U_j^c)+P(B)+P(\tilde{B})+P(\bigcap_{\ell=0}^{t'^{\frac{m-1}{2m-1}}/3} C_\ell)\\
&\leq P(\bigcap_{j:\zeta_j^h\in[s',s'+t']} U_j^c)+P(B)+P(\tilde{B})+P(\bigcup_{\ell=0}^{t'^{\frac{m-1}{2m-1}}/3} \hat{C}_\ell^c)+P(\bigcap_{\ell=0}^{t'^{\frac{m-1}{2m-1}}/3} C_\ell\cap\hat{C}_\ell)\\
&\leq P(\bigcap_{j:\zeta_j^h\in[s,s+t]} U_j^c)+P(B)+P(\tilde{B})+\sum_{\ell=0}^{t'^{\frac{m-1}{2m-1}}/3} P(\hat{C}_\ell^c)+(P(C_\ell\cap\hat{C}_\ell))^{t'^{\frac{m-1}{2m-1}}/3}\\
&\leq e^{-c_1t'}+e^{-c_3t'^{\frac{m}{2m-1}}}+e^{-c_6t'}+e^{-c_2t'^{\frac{m}{2m-1}}}+(A_m\exp(-\bar{a}_m t'^{1/(2m-1)}))^{t'^{\frac{m-1}{2m-1}}/3}\\
&\leq \bar{A}'_m\exp(-\bar{b}'_m t'^{\frac{m}{2m-1}})
\end{aligned}$$

for large t', where A'_m and b'_m are some positive constant.

So we know the statement in (2.49) holds for all $\theta\geq\frac{2m-1}{m}$. Start with $m_1=2$, we have a recursion equation $m_k=\frac{2m_{k-1}-1}{m_{k-1}}$ and we know (2.49) holds for all $\theta\geq m_k$. It is not hard to see that $m_k=\frac{k+1}{k}$ and thus $\lim_{k\to\infty}m_k=1$. Now observe that for $m_k=\frac{k+1}{k}$, we have $t'^{\frac{m_k}{2m_k-1}}>\sqrt{t'}$ for $k>1$.

So, for some constant $\bar{a}_\theta$ which is a function of Δ', we can rewrite (2.49) as

$$\tilde{q}[\alpha(s),\alpha(s+t)]\leq\bar{A}'_m\exp(-\bar{a}_\theta t^{1/\theta})$$

which concludes the lemma.

2.8.7 Proof of Lemma 9

Let U_j be the event in $ss1$ that the $j-$th honest block b_j is a loner, i.e.,

$$U_j=\{\tau_{j-1}^h<\tau_j^h-\Delta\}\bigcap\{\tau_{j+1}^h>\tau_j^h+\Delta\}$$

Let $\hat{F}_j=U_j\bigcap F_j$ be the event that b_j is a Nakamoto block. We define the following "potential" catch up event in $ss1$:

$$\hat{A}_{ik}=\{D_i(\alpha(\tau_k^h+\Delta))\geq D_h(\alpha(\tau_{k-1}^h))-D_h(\alpha(\tau_i^h+\Delta))\},\tag{2.53}$$

which is the event that the adversary launches a private attack starting from honest block b_i and catches up the fictitious honest chain right before honest block b_k is proposed.

Next, define the following events

$$V_j^{ss1}=\{\alpha(\tau_{j-1}^h)<\alpha(\tau_j^h)-\frac{\lambda_{\max}}{\lambda_h}\Delta\}\bigcap\{\alpha(\tau_{j+1}^h)>\alpha(\tau_j^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta\}\tag{2.54}$$

$$\hat{B}_{ik}^{ss1}=\{D_i(\alpha(\tau_k^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta)\geq D_h(\alpha(\tau_{k-1}^h))-D_h(\alpha(\tau_i^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta)\}\tag{2.55}$$

Lemma 14. *For any pair of i,k,*

$$\hat{A}_{ik}\subseteq\hat{B}_{ik}^{ss1}.$$

Proof. Using equation 2.9, we have

$$\begin{aligned}
\alpha(\tau_k^h+\Delta)&=\int_0^{\tau_k^h+\Delta}\frac{\lambda_h^c(u)}{\lambda_h}du=\int_0^{\tau_k^h}\frac{\lambda_h^c(u)}{\lambda_h}du+\int_{\tau_k^h}^{\tau_k^h+\Delta}\frac{\lambda_h^c(u)}{\lambda_h}du\\
&\leq\alpha(\tau_k^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta
\end{aligned}$$

Similarly, $\alpha(\tau_i^h+\Delta) \leq \alpha(\tau_i^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta$. Because $D_h(.)$ and $D_i(.)$ are increasing functions over their domain, we have

$$D_i(\alpha(\tau_k^h+\Delta)) \leq D_i(\alpha(\tau_k^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta) \text{ and}$$
$$D_h(\alpha(\tau_i^h+\Delta)) \leq D_h(\alpha(\tau_i^h)+\frac{\lambda_{\max}}{\lambda_h}\Delta)$$

□

Lemma 15. *For all j,*

$$V_j^{ss1} \subseteq U_j.$$

Proof. This can be proved using the fact that $\int_{\tau_{j-1}^h}^{\tau_{j-1}^h+\Delta} \frac{\lambda_h^c(u)}{\lambda_h}du \leq \frac{\lambda_{\max}}{\lambda_h}\Delta$ and $\int_{\tau_j^h}^{\tau_j^h+\Delta} \frac{\lambda_h^c(u)}{\lambda_h}du \leq \frac{\lambda_{\max}}{\lambda_h}\Delta$. □

By time-warping, R_m is an IID exponential random variable with rate λ_h. Let $\zeta_j^h = \alpha(\tau_j^h)$, that is, ζ_j^h is the time of mining of $j-$th honest block in the local clock of static system $ss1$. Similarly, we define $\zeta_j^a = \alpha(\tau_j^a)$ for the $j-$th adversarial block. Then, we can rewrite the event $\hat{B}_{ik}$ as:

$$\hat{B}_{ik}^{ss1} = \left\{ D_i(\zeta_k^h+\frac{\lambda_{\max}}{\lambda_h}\Delta) \geq D_h(\zeta_{k-1}^h) - D_h(\zeta_i^h+\frac{\lambda_{\max}}{\lambda_h}\Delta) \right\}.$$

Lemma 16. *In the static system $ss1$, for each j*

$$P(\hat{F}_j^c) = P(F_j^c \cup U_j^c) \leq P\left(\left(\bigcup_{(i,k):0\leq i<j<k} \hat{B}_{ik}^{ss1} \right) \cup (V_j^{ss1})^c \right). \tag{2.56}$$

This can be proved in a similar way as Lemma 12 and using Lemma 14, 15. Furthermore, defining X_d, $d>0$, as the time it takes in the local clock of static system $ss1$ for D_h to reach depth d after reaching depth $d-1$, we have

Proposition 7. *Let Y_d, $d \geq 1$, be i.i.d random variables, exponentially distributed with rate λ_h. Then, each random variable X_d is less than $\Delta' + Y_d$, where $\Delta' = \frac{\lambda_{\max}}{\lambda_h}\Delta$.*

Proof. Let h_i be the first block that comes at some depth $d-1$ within $\mathcal{T}_h$. Then, in the local clock of static system, every honest block that arrives within interval $[\alpha(\tau_i^h), \alpha(\tau_i^h+\Delta)]$ will be mapped to the same depth as h_i, i.e., $d-1$. Hence, $\mathcal{T}_h$ will reach depth d only when an honest block arrives after time $\alpha(\tau_i^h+\Delta)$. Now, due to time warping, in the local clock of static system $ss1$, we know that the difference between $\alpha(\tau_i^h+\Delta)$ and the arrival time of the first honest block after $\alpha(\tau_i^h+\Delta)$ is exponentially distributed with rate λ_h due to the memoryless property of the exponential distribution. This implies that for each depth d, $X_d = \alpha(\tau_i^h+\Delta) - \alpha(\tau_i^h) + Y_d = \int_{\tau_i^h}^{\tau_i^h+\Delta} \frac{\lambda_h^c(u)}{\lambda_h}du + Y_d \leq \Delta' + Y_d$ for some random variable Y_d such that $Y_d, d \geq 1$, are IID and exponentially distributed with rate λ_h. □

Thus, for $\Delta' = \frac{\lambda_{\max}}{\lambda_h}\Delta$, Proposition 7 implies that both Proposition 4 and Proposition 6 are satisfied for the static system $ss1$. Therefore, for $\Delta' = \frac{\lambda_{\max}}{\lambda_h}\Delta$, a similar result holds for the event $\hat{B}_{ik}^{ss1}$ as in Lemma 13. Additionally, Lemma 6 holds for $ss1$. Then, substituting $\Delta' = \frac{\lambda_{\max}}{\lambda_h}\Delta$ and using Lemma 16, we have both Lemma 7 and Lemma 8 satisfy for the static system $ss1$. For a time $t>0$ in the local clock of the dynamic available system $dyn2$, we have $\alpha(t) \geq \frac{\lambda_{\min}}{\lambda_h}t$. Then, using Lemma 3, Lemma 4, Lemma 5, we conclude the proof.

Algorithm 2 PoSAT

```
1: procedure INITIALIZE                                              ▷ all variables are global
2:     blkTree ← SYNC()                                              ▷ syncing with peers
3:     unCnfTx ← φ                                                   ▷ pool of unconfirmed txs
4:     parentBlk ← blkTree.TIP()                                     ▷ tip of the longest chain in blkTree
5:     randSource ← None                                             ▷ will be updated at next epoch beginning
6:     slot ← None                                                   ▷ will be updated at next epoch beginning
7:     return False
8: procedure POSLEADERELECTION(coin)
9:     (RANDVDF.ek, RANDVDF.vk),(SIGN.vk, SIGN.sk) ← coin.KEYS()
10:    stake ← coin.STAKE( SEARCHCHAINUP(parentBlk))                 ▷ update the stake
11:    s ← UPDATETHRESHOLD(stake)                                    ▷ update the threshold
12:    input ← randSource
13:    // Calling RANDVDF.EVAL
14:    (input,output,proof,randIter,slot) ← RANDVDF.EVAL(input,ek,s,slot)
15:    randSource ← output                                           ▷ update source of randomness
16:    state ← HASH(parentBlk)
17:    content ← ⟨unCnfTx,coin,input,randSource,proof,randIter,state,slot⟩
18:    return ⟨header,content,SIGN(content,SIGN.sk)⟩
19: procedure RECEIVEMESSAGE(X)                                      ▷ receives messages from network
20:    if X is a valid tx then
21:        unCnfTx ← unCnfTx ∪ {X}
22:    else if ISVALIDBLOCK(X) then
23:        if parentBlk.LEVEL() < X.LEVEL() then
24:            CHANGEMAINCHAIN(X)                                    ▷ if the new chain is longer
25:            parentBlk ← X                                         ▷ update the parent block to tip of the longest chain
26:            if X.LEVEL() % c == 0 then
27:                randSource ← X.content.randSource
28:            else
29:                randSource ← randSource
30:            if participate == True then
31:                RANDVDF.RESET()                                   ▷ reset the RANDVDF
32:            // Epoch beginning
33:            if (X.LEVEL() % c == 0) & (participate == False) then
34:                slot ← X.content.slot
35:                participate = True
36: procedure ISVALIDBLOCK(X)                                        ▷ returns true if a block is valid
37:    if not ISUNSPENT(X.content.coin) then return False
38:    if PARENTBLK(X).content.slot ≥ X.content.slot then
39:        return False                                              ▷ ensuring time ordering
40:    s ← UPDATETHRESHOLD(PARENTBLK(X))
41:    if HASH(X.content.{randSource,slot}) > THRESHOLD(s) then return False
42:    // verifying the work
43:    return RANDVDF.VERIFY(X.coin.vk,X.content.{input,randSource,proof,randIter})
44: procedure MAIN( )                                                ▷ main function
45:    participate = INITIALIZE()
46:    STARTTHREAD(RECEIVEMESSAGE)                                   ▷ parallel thread for receiving messages
47:    while True do
48:        if participate == True then
49:            block = POSLEADERELECTION(coin)
50:            SENDMESSAGE(block)                                    ▷ broadcast to the whole network
```

Chapter 3

Hammurabi : Order-Fair Consensus in the Permissionless Setting

Transaction-order-manipulation attacks have become extremely common in public blockchains such as Ethereum, costing hundreds of millions of dollars. In these blockchains, a miner can unilaterally determine the order of transactions inside a block, and this ordering is not checked by other users, leaving room for the miner to manipulate the order for its own benefit. This gap is also evident from existing security results for permissionless blockchains. As prime examples, the breakthrough work of Garay et al. (Eurocrypt 2015) and Pass et al. (Eurocrypt 2017) showed the security properties of *consistency* and *liveness* for Nakamoto's seminal proof-of-work protocol. However, consistency and liveness do not provide any guarantees on the relationship between the order in which transactions arrive into the network and the finalized order in the ledger.

As a solution, a recent paper by Kelkar et al. (Crypto 2020) introduced a third useful property for consensus protocols: *(transaction)-order-fairness*, which proposes a strong relationship between the transaction arrival order and their order in the ledger. Their model was limited to the classical (permissioned) setting however, where the set of protocol nodes is fixed a priori, and does not fit well for permissionless environments where order-manipulation attacks have been most prominent. In this work, we initiate the investigation of order-fairness in the permissionless setting and design two protocols that realize this new property addition to standard requirements of consistency and livenesss. The key insight behind our protocols in providing order-fairness is that a miner can no longer unilaterally determine ordering and proposals from many miners are combined in a fair way to construct the finalized ordering. As a simple but unexpected corollary, we show that any fair ordering protocol achieves a powerful zero-block confirmation property, through which honest transactions can be securely confirmed *even before they are included in any block*.

3.1 Introduction

Transaction-order-manipulation attacks. Decentralized financial (DeFi) systems have become massively popular on permissionless blockchains in the last few years. As of May 2021, the Ethereum blockchain houses more than 75 billion USD [11] of locked capital in DeFi smart contracts. On Ethereum, popular lending contracts like Maker [12] and Compound Finance [7], and decentralized exchanges (DeXes) like Uniswap V2 [14] and Curve [9] each routinely handle hundreds of millions of dollars in transaction volume every day [10]. With this kind of volume, transaction ordering is crucially important, as the execution order can determine the validity and profitability of a transaction. In many scenarios, *when* a transaction is executed is even more important than *whether* it was executed at all. Execution in an unfavorable order can result in unnecessary losses for the user, for example, by requiring a higher collateral (for lending contracts) or by receiving a worse token exchange rate (in DeXes). More problematically, adversarial manipulation of the transaction ordering has been shown to be extraordinarily lucrative for the attacker [63, 142, 169]. In fact, attackers have been seen to

compete with each other in an attempt to finalize the ordering most profitable to them [63].

This has made (transaction)-order-manipulation attacks extremely common in public blockchains such as Ethereum, costing hundreds of millions of dollars [161, 139, 129], and recently also resulting in the now infamous *Black Thursday* event [35]. In DeFi systems, order-manipulation attacks have been shown to result in adversaries extracting profit at the expense of ordinary users [63, 75], systemic bribery [123], and even fundamental protocol instability [63, 108]. Recently, the proposed Ethereum improvement proposal (EIP-1559) [2], which cuts transaction fees sent to the miner, has exacerbated the problem of order-manipulation by incentivizing miners to add order-manipulation software [57] or auction of the right to order in their block [80, 83].

Order-manipulation attacks are enabled because a miner can unilaterally determine the order of transactions inside a block, and this ordering is not checked by other users, leaving room for the miner to manipulate the order for its own benefit. Such unilateral manipulation points to a core fundamental issue in blockchains: there is no relationship between the order in which transactions arrive into the network and the final transaction ordering in the log. Neither consistency nor liveness—the two pillars of distributed-system security—enforces any guarantees on the relationship between the order in which transactions arrive into the network and the final transaction ordering in the log.

Order-Fair Consensus. As a solution, recent work by Kelkar, Zhang, Goldfeder, and Juels [104] (henceforth KZGJ) proposed a new consensus property, called *order-fairness*. Informally, if a transaction tx_1 was received before another one tx_2 by a large fraction of the nodes in the network, order-fairness dictates that tx_1 be sequenced in the output log first. While prior work in classical distributed systems [143, 41] had considered a limited notion of preventing reordering based on transaction content, order-fairness is able to handle any kind of adversarial censorship, reordering, and transaction insertions, even those based on user collusion. However, KZGJ only considered the permissioned setting. Consequently, the protocols in KZGJ do not directly apply to previously mentioned attacks which were largely on permissionless blockchains. Our work extends the work of KZGJ to the permissionless setting. We compare to other related works in Section 3.7.

Defining order-fairness in the permissionless setting (Section 5.2). We introduce the first formalization of a fair transaction ordering property in the permissionless setting by generalizing *order-fairness* for the permissioned setting from KZGJ . In the permissioned setting, where the n system nodes were fixed, order-fairness was defined in terms of being received by a large fraction (parameterized by γ) of nodes. For the permissionless setting however, since new nodes may join the network, it does not make sense to consider a fraction of all nodes that ever existed. Furthermore, it is not relevant to consider the transaction input ordering of a node that spawned much later either. Instead, to determine the ordering of a transaction, we will look at the system at the time when the transaction was initially propagated through the network.

Permissionless order-fairness is impossible to achieve and not particularly interesting in the presence of completely dynamic adversaries. For instance, such an adversary can quickly corrupt and kill all the nodes that were online during the initial propagation of a transaction, before they have a chance to mine any blocks. Any new honest nodes that spawn later will not retain the transaction ordering received by earlier nodes. Therefore, we will need to restrict our adversaries to be (τ,R)-respawning, i.e., in any R round period, at most τ fraction of nodes can be killed (and respawned). We still allow the adversary to corrupt nodes instantly.

We find that extrapolating the network formalism from KZGJ to the permissionless setting brings our several subtleties. KZGJ considered a non-standard network model, that allowed for a cleaner definition of order-fairness. But while the modeling was reasonable in a permissioned network, it does not fit real-world permissionless systems. We discuss our modeling choices and their subtleties extensively in Section 5.2 and Appendix 3.8.

Key insight. As mentioned before, in existing blockchains, when a block is mined by an adversarial miner, the miner unilaterally orders the transactions inside the block according to its own preferences (or possibly even as chosen by an external briber). The key idea we use to get around this unilateral

proposition of ordering of transactions is the following: ordering for each transaction (w.r.t. other transactions) is proposed across multiple, say d, blocks. The proposed orderings within these blocks are then aggregated in a *fair* way to obtain the final transaction ordering. Each miner reports their own preferences on how to order a set of transactions — and honest nodes will report their true received order of transactions from the peer-to-peer network, while adversarial nodes can arrange transactions in arbitrary order in the block it mines so as to influence the final ordering obtained from consensus. Owing to network delay or adversarial behavior in the network, different nodes might receive the same set of transactions from the p2p network in different orders and at different times. These differences will get reflected in the order in which the transactions are arranged in each of the d orderings that were used for reaching consensus on their final ordering. e call this design structure Hammurabi .

In essence, the system as a whole will mine d independent transaction orderings, which will be appended to as new transactions arrive. We call these orderings "semantic chains" since transactions in the same list are logically or semantically connected even though they may directly follow a different mined block. All transactions will be mined d times, once in each semantic chain. The final ordering will be extracted from the d semantic chains. Notably, d is a function only of the security parameter κ and not of the number of nodes. The upshot of such an approach is that in our protocols, the adversary will not be able to manipulate the ordering within a majority of semantic chains, resulting in a fair overall ordering. Figure 3-1 highlights this key idea as compared to existing protocols.

Fair extraction (Section 3.4). We detail how to extract an ordering from the semantic chains in a way that satisfies order-fairness. Specifically, we introduce a novel streaming finalization problem that formulates the extraction of a fair ordering from d transaction ordering lists $\mathcal{L} = [\mathsf{List}_1, ..., \mathsf{List}_d]$ that are provided as input in a streaming fashion.

To solve the streaming finalization problem, we start by distilling the key techniques from the permissioned Aequitas protocol from [104] into an algorithm. Here, care must be taken since in the permissionless setting, all orderings can have adversarial influence, whereas in the simple permissioned setting, a majority of orderings were completely honest. To get around this, we show a player replaceability lemma that allows us to represent the semantic chains as transaction orderings of d "virtual" nodes in a permissioned setting, as long as a natural completeness property is satisfied by the way the semantic chains are built. This effectively allow us to reduce the permissionless order-fairness problem to its permissioned analogue by constructing ledgers held by virtual nodes. Along the way, we highlight interesting connections to voting theory that may be of independent interest.

Permissionless protocols for order-fairness (Section 3.3). Starting from the pioneering idea in [128], longest-chain based protocols have been constructed for many different permissionless settings: proof-of-work [91, 132], proof-of-stake [62, 107, 66] and proof-of-space [146]. Following our Hammurabi design, we provide two different constructions in the synchronous setting that modify any longest-chain based protocol and endow it with the order-fairness property. While our constructions are generic to any longest-chain protocol, we describe them in the context of proof-of-work (PoW) for concreteness. Both our constructions follow the same basic outline. First, through PoW mining, nodes mine blocks, and from the blockchain, one can deduce d distinct transaction-order lists or semantic chains. We then apply our fair extraction algorithm $\mathsf{Aequitas}(\cdot)$ to retrieve the fair ordering.

For both these protocols, we characterize the adversarial threshold below which they achieve order-fairness in the PoW setting. Since our proofs are built on basic chain-quality and growth properties of the underlying blockchain, the proofs can be extended to the other permissionless settings such as proof-of-stake and proof-of-space as well. We briefly describe our constructions below.

Single-Chain protocol Π_{mod} (Section 3.3.1). Our first protocol uses a single PoW Nakamoto chain. Here, the d semantic chains will arise from the chain *modulo* d. Specifically, blocks at indices that are congruent modulo d will become part of the same semantic or *modular* chain. This allows any node to construct the d lists from the public blockchain and run $\mathsf{Aequitas}(\cdot)$ to retrieve the final fair ordering. Assuming (τ, R)-respawning adversaries that corrupt at most β fraction of nodes, for a sufficiently large R, we show that Π_{mod} satisfies order-fairness when $\beta + \tau < \frac{1}{3}$. Consistency is still maintained as long as $\beta < \frac{1}{2}$. We also show in Section 3.10.1 that using an underlying mining-fair chain like Fruitchains [133] instead, we can tolerate $\beta + \tau < \frac{1}{2}$ for order-fairness.

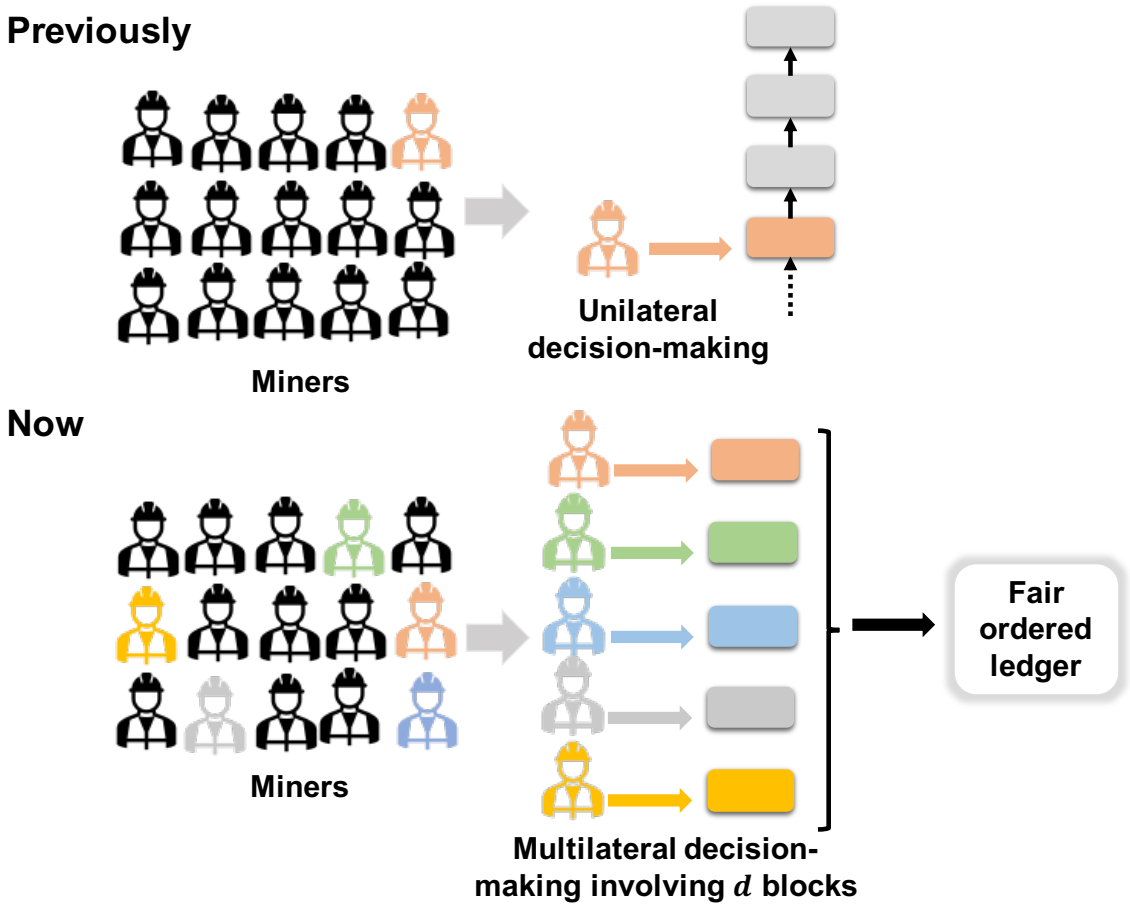


Figure 3-1: Hammurabi Order-Fair Protocol Design

Multi-Chain protocol Π_{multi} (Section 3.3.2). Our second protocol, Π_{multi}, has a more parallel design for the semantic chains, with the d PoW chains being mined simultaneously. It achieves lower latency than our single-chain protocol, and more importantly, it presents a new method to combine multiple independent chains in a fair way. For this protocol, we use the d-for-1 PoW mining technique modified from [91, 133, 28] to determine which chain a block gets mined into. Once again, we can now run $\mathsf{Aequitas}(\cdot)$ to retrieve the final fair ordering. Assuming (τ,R)-respawning adversaries that corrupt at most β fraction nodes, we show order-fairness as long as $\beta+\tau<\frac{7-\sqrt{17}}{16}\approx 0.1798$ and consistency as long as $\beta<\frac{1}{2}$. The non-triviality in obtaining this result is primarily in ensuring that an adversary gains no head-start over honest nodes by privately mining a transaction onto several chains before releasing the mined blocks at once. The proof establishes a novel cross-chain-quality property using techniques from the queuing theory literature and may be of independent interest to the consensus community.

Zero-block confirmation and latency reduction (Section 3.6). We show a surprising but simple result here that in any fair ordering protocol, an honest transaction can be locally finalized very fast, *within two-times the network delay*. We note that while there is a large body of work trying to improve the confirmation latency of permissionless systems [82, 166, 114, 28], none of them are able to achieve fast confirmation at this time-scale, since all of them require mining at least a certain number of blocks at a much slower rate than network delay. We call this surprising but simple property as *zero-block confirmation*. Suppose that an honest node sees tx in round r and does not see a conflicting transaction within 2Δ. Then the honest node can immediately infer that tx will be ordered ahead of any conflicting transaction in the final ledger and can finalize tx. We note that the node may still need

to wait much longer (for blocks to be mined) to know the precise location in the final ledger.

Practitioners have previously used similar techniques [100] for quickly confirming transactions. Indeed, potential zero-block-confirmation has been a topic of interest since the early days of Bitcoin, especially as it makes everyday real-world purchases (e.g., buying a coffee) with Bitcoin more practical (by not requiring a confirmation time of minutes or hours). However, in Bitcoin, it can happen that a double-spend transaction appearing much later than the original transaction can still precede it in the finalized ledger (e.g., as a result of an adversarial miner). In contrast, in a fair-ordered protocol, this is not possible, thus making zero-block confirmation after waiting for 2Δ time secure.

We emphasize that although zero-block-confirmation follows easily from order-fairness, current permissionless protocols, even ones in the synchronous setting, do not provably achieve such a property, and that it is not a direct artifact of network synchrony.

3.2 Model and Preliminaries

We describe the general execution model in this section. Much of our formalism is adapted from an extensive line of research on permissionless consensus [91, 133, 132, 135, 137]. We also use some of the formalism related to order-fairness from KZGJ [104]. Lifting the network and order-fairness formalism from KZGJ to the permissionless setting brings out several subtleties which we detail in Sections 3.2.1 and 3.2.3 and Appendix 3.8.

General Execution Model. As standard practice [91, 132], we model protocol nodes as ITMs [48], and use a special environment machine $\mathcal{Z}$ to direct execution. We assume a synchronous model where execution proceeds in *rounds*. At the start of a round, nodes receive transactions as input from $\mathcal{Z}$; at the end of a round, nodes may deliver[1] outputs to $\mathcal{Z}$. For ease of modeling, for honest nodes, the final ordering will be taken as the result of a transformation function $\mathsf{linearize}(\cdot)$ (part of the protocol description) on the outputs to $\mathcal{Z}$ rather than the outputs themselves. This will extract the relevant information from the outputs into a total ordering. Similar formalism is present in e.g., [137]. New transactions are modeled as being input by $\mathcal{Z}$ to avoid explicitly having to model clients.

Corruptions. $\mathcal{Z}$ can spawn new nodes either as *corrupt* or *honest*. Honest nodes execute the protocol faithfully while corrupt nodes are controlled by an adversary $\mathcal{A}$, and can deviate arbitrarily. Existing nodes can also be corrupted, which gives $\mathcal{A}$ full access to them. $\mathcal{A}$ can kill corrupted nodes which removes them from execution.

We observe that if the adversary is able to corrupt and kill nodes at will, it can cause the network to "forget" the transaction input orderings received by earlier honest nodes. While this was not important for prior work, this is critically detrimental for any fair ordering protocol. Therefore, we will assume a *(mildly)-respawning-adversary* that limits the node churn. Informally, a (τ,R)-respawning[2] adversary can kill at most τ fraction nodes within any period of R rounds. This corresponds to the notion that $\mathcal{A}$ cannot quickly corrupt and kill many nodes. We will distinguish between the respawning parameter and the adversarial corruption parameter (see Definition 11).

Notational conventions. κ denotes the security parameter. For a protocol Π, $\mathsf{EXEC}^{\Pi}(\mathcal{A},\mathcal{Z},\kappa)$ represents the random variable for all possible execution traces of Π w.r.t. $(\mathcal{A},\mathcal{Z})$. The possible executions arise from any randomness used by honest nodes, adversarially controlled nodes, and the environment. Any view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A},\mathcal{Z},\kappa)$ is a fully specified instance (including inputs, outputs, random coins etc.) of an execution trace.

[1] In this paper, we will use the terms *deliver* and *output* interchangeably.

[2] This is distinguished from an R-mild adversary in [134] where the adversary takes R rounds to *corrupt* a node. While the security properties of [134] crucially relied on the adversary not gaining control of nodes instantly, we can tolerate the adversary corrupting a node instantly, as long as it can't kill too many nodes quickly.

3.2.1 Communication Networks

Standard formalism for consensus protocols models a single communication network amongst consensus nodes. The formalism from KZGJ , however, models two separate networks: an external network through which clients send messages, and an internal network for node communication. The adversary is assumed to only be in charge of the internal network. This was important for them to meaningfully discuss fair ordering, as it allowed honest nodes to have an unmolested view of their local transaction input ordering. This modeling is less realistic in the permissionless setting however, where nodes usually communicate through a gossip-style network. Therefore, we resort to modeling a single network that in spirit, melds together both networks from KZGJ . This brings out several subtleties in the network modeling, and we defer the detailed description to Appendix 3.8. We directly proceed with the formal network definition we use below.

Network Formalism. We model a single communication network that in spirit melds the external and internal network from KZGJ . We assume a synchronous model of communication, such that messages sent are delivered within a (known) bounded number of rounds Δ (a function of κ). The adversary $\mathcal{A}$ can delay and reorder messages (subject to the bound Δ) but cannot drop messages. We define the synchrony assumption below.

Definition 10 (Δ-synchrony)**.** *$(\mathcal{A},\mathcal{Z})$ respects Δ-synchrony if for all* view *in the support of* $\mathsf{EXEC}^{\Pi}(\mathcal{A},\mathcal{Z},\kappa)$*, the following hold:*

- *(Input Synchrony) If an honest node receives* tx *as input from $\mathcal{Z}$ in round r, then in any round $r' \geq r+\Delta$, any honest node in the system at round r' will have already received* tx *as input from $\mathcal{Z}$. This includes any node that may have spawned in round r'.*
- *(Ledger Synchrony) If an honest node outputs* tx *to $\mathcal{Z}$ in round r (recall that this is not the same as being ordered in the final ledger), then in any round $r' \geq r+\Delta$, any honest node in the system in round r' will have already received* tx *as input from $\mathcal{Z}$. This includes any node that may have spawned in round r'.*
- *(Broadcast Synchrony) If an honest node sends a message in round r, then in any round $r' \geq r+\Delta$, any honest recipient node will have received the message by round $r+\Delta$. This includes any node that may have spawned in round r'.*
- *$\mathcal{Z}$ provides Δ to all nodes when they are spawned.*

Remark 2 (Input Synchrony and DoS)**.** *Similar to the external synchrony assumption in* KZGJ *, input synchrony assumes that the network gossips all transactions. In a permissionless network, this could lead to a denial-of-service (DoS) attack wherein an adversary floods the network with bogus double-spends (e.g., transactions that spend the same tokens). The Bitcoin network mitigates this by not gossiping any double-spends. This can be done for our protocols as well. We emphasize that doing so does not actually invalidate the order-fairness property of our protocols since we only require adversarial transactions mined in a PoW chain to be gossiped.*

We also note further that the proposal in [100] to prevent fast double-spends (although it cannot guarantee that later double-spends are not sequenced earlier due to adversarial miners) involves gossiping all double-spends through the entire network, for which the authors argue that the performance penalty incurred by the network would not be significant.

3.2.2 Permissionless Formalism

We recall the definition of the permissionless setting next, taken from [137], and add to it our respawning parameters.

Definition 11 ((n,β,Δ,τ,R)-permissionless environments)**.** *We say that $(\mathcal{A},\mathcal{Z})$ respects (n,β,Δ,τ,R)-permissionless execution w.r.t. a protocol Π if for every $\kappa \in \mathcal{N}$, $(\mathcal{A},\mathcal{Z})$ respects Δ-synchrony, and for*

all view *in the support of* $\mathsf{EXEC}^{\Pi}(\mathcal{A},\mathcal{Z},\kappa)$*, the following holds: (1) In each round, there are exactly*[3] *n nodes online in the system, of which at most* βn *are corrupt; (2) Within any period of* R *rounds, at most* τn *notes are killed. We call such an adversary* (τ,R)*-respawning; (3)* $\mathcal{Z}$ *provides all nodes the parameters* n,β,Δ,τ,R *upon spawning. Note that* n,Δ,R *are functions of* κ*.*

Blockchain formalism. An abstract blockchain [132] chain is an ordered sequence of blocks output by an honest node. Let $\mathsf{chain}[i]$ denote the vector of transactions contained in chain at index i. For a given chain and non-negative integers b and b', we use $\mathsf{chain}[b]$ to denote the b^{th} block of chain (indexed from 0), $\mathsf{chain}[:b]$ to denote the first b blocks, and $\mathsf{chain}[b:b']$ to denote the sequence of blocks from $\mathsf{chain}[b]$ to $\mathsf{chain}[b']$, both inclusive. We also use $\mathsf{chain}[-b]$ $(b\neq 0)$ to denote the b^{th} block from the rear and $\mathsf{chain}[:-b]$ to denote chain except for the last b blocks.

We assume familiarity with the (by now) standard requirements for blockchain protocols, and refer the reader to [91, 132, 137] for more details. Informally, for honest chains, we require (1) T-common-prefix which allows confirmation of all except the last T blocks; (2) (T,g_0,g_1)-chain-growth which signals a growth of at least T in T/g_0 rounds and at most T blocks in T/g_1 rounds; (3) (T,μ)-chain-quality, which means that within any T consecutive blocks, at least μT are mined by honest nodes. Our proofs will require these properties from previous well-studied constructions only in a black-box way. For completeness, we also give the details in Appendix 3.8.2.

Abstract blockchain protocols directly satisfy SMR consistency and livenes (see [132, 137]). For our purpose, we define $(T_{\text{warmup}},T_{\text{confirm}})$-liveness as follows: If an honest node is input m by $\mathcal{Z}$ in round $r\geq T_{\text{warmup}}$, then for any $r'\geq r+T_{\text{confirm}}$ and any honest node in round r' that outputs $\mathcal{X}$ to $\mathcal{Z}$, m is in $\mathsf{linearize}(\mathcal{X})$. When there is no warmup time, we simply write T_{confirm}-liveness.

Random Oracle. We assume that all nodes have access to a random function $H:\{0,1\}^*\to\{0,1\}^\kappa$. This is provided by two oracles: $\mathsf{H}(x)$ which outputs $H(x)$ and $\mathsf{H.ver}(x,y)$ which outputs 1 if $H(x)=y$ and 0 otherwise. In any round, all parties can make q queries to H and any number of queries to $\mathsf{H.ver}$. An adversary $\mathcal{A}$ controlling p parties is allowed to make pq sequential oracle queries. Note that $\mathcal{Z}$ cannot access H, but can instruct $\mathcal{A}$ to make queries. This abstraction follows [91, 132], and captures the intuition that finding a proof-of-work solution is difficult, while checking the validity of a solution is inexpensive.

Blocks. A block $\mathtt{B}$ is a tuple $(h_{-1},h',\mathtt{m},\eta,h)$, where h_{-1} denotes a pointer to a parent block, h' denotes a pointer to a reference block (an arbitrary previous block), $\mathtt{m}$ denotes the block record (eg., a vector of transactions), η denotes the PoW nonce, and h denotes the hash of the current block. The reference block is an important modeling choice since it allows $\mathtt{B}$ to *logically* follow a different block than its preceding parent block. Notably, this allows the validity condition for a block to also be defined in terms of the reference block, and possibly separately from the parent block. We note that $h_{-1}=h'$ in many cases (eg., the Nakamoto protocol).

Protocols are parameterized by a hardness function $p(\cdot)$ that defines $D_p=p(\kappa)\cdot 2^\kappa$ such that $\Pr_\eta[H(h_{-1},h',\mathtt{m},\eta)<D_p]=p(\kappa)$ for all $(h_{-1},h',\mathtt{m})$. Now, we define a valid block $\mathtt{B}$ as a tuple $(h_{-1},h',\mathtt{m},\eta,h)$ where $H(h_{-1},h',\mathtt{m},\eta)=h$ and $h<D_p$. We use $\Pi_{\text{nak}}(p)$ to denote Nakamoto's protocol and $\Gamma^p_{\text{nak}}(n,\beta,\Delta)=1$ to denote a Nakamoto-compliant [132] execution. For completeness, we also provide the details in Appendix 3.8.2.

3.2.3 Order-Fairness

Background on order-fairness. An extensive number of transaction reordering and front-running attacks [122, 63, 75], have been observed recently on real-world blockchains; in particular, in the context of Decentralized Apps (or DApps) on Ethereum. While some prior protocols have attempted to provide partial fixes for ordering attacks, KZGJ initiated the first formal study for fairness of transaction ordering in the permissioned setting. Informally, KZGJ defines γ-receive-order-fairness (resp.

[3]While we model a fixed n in each round similar to most blockchain papers, we note that our model can easily be extended to handle a varying n, using complementary techniques [89, 52] that are well understood in literature.

γ-batch-order-fairness) as the following property: If γ fraction of nodes receive as input a transaction tx before another one tx$'$, then tx should be included before (resp. no later than) tx$'$ in the final ordering. We emphasize (as shown in KZGJ) that receive-order-fairness is a strictly stronger ordering property than previously considered notions like censorship resistance, random leader election, threshold encryption etc.

While KZGJ studied fair transaction ordering in the permissioned setting, since almost all of the aforementioned real-world attacks are on permissionless systems, it is important to extend the order-fairness definition to the permissionless setting. We will do exactly that in this section. In Section 3.7, we compare our fairness definition with other related ordering notions.

Choosing an order-fairness definition in the permissionless setting. Consider two transactions tx and tx$'$. First, we note that in the permissionless setting, nodes that are spawned far later (eg., after tx and tx$'$ are output), will receive tx and tx$'$ in the same round (see the input synchrony definition). Consequently, we cannot naïvely use the same order-fairness definition from the permissioned setting, since the antecedent will never be satisfied making the definition vacuous. Instead, we only consider the first time that the n nodes in the system receive either transaction, corresponding to the initial propagation of the transactions.

Let $\mathcal{S}$ be the set of these nodes. We say that a node has received tx before tx$'$ if it has either (1) received tx but not tx$'$ or (2) received both transactions and received tx in an earlier round than tx$'$. Now, if a large fraction (parameterized by γ) of nodes in $\mathcal{S}$ have received tx before tx$'$, then intuitively, tx should be preferred over tx$'$. For receive-order-fairness (resp. batch-order-fairness), this means that the final log will contain tx before (resp. no later than) tx$'$. Henceforth, for brevity, when we say that the final log contains tx before tx$'$, we also include the scenario where the log only contains tx. Permissionless order-fairness is formalized in Definition 12.

Definition 12 (Order-Fairness)**.** *For a given* view *in the support of* $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$*, we define* γ*-receive-order-fairness (resp.* γ-batch-order-fairness *) as follows: For two transactions* tx *and* tx$'$*, let* r *be the first round such that at least* n *nodes alive in round* r*, have received either* tx *or* tx$'$*. Now, if* γ *fraction of those nodes have received* tx *before* tx$'$ *as input, then for any round* $t \geq r$*, and any node* i *that is honest in round* t *and outputs* $\mathcal{X}$*, the final linearly ordered log,* $\mathsf{linearize}(\mathcal{X})$*, either (1) contains neither transaction; (2) contains only* tx*; or (3) contains* tx *before (resp. no later than)* tx$'$*.*

We say that a protocol satisfies order-fairness if it satisfies the above, except with negligible probability over a random view*.*

Similar to KZGJ , the batch relaxation is used since KZGJ found that receive-order-fairness is impossible to achieve except in very specific settings. Note that for batch-order-fairness , transactions within a batch can still be totally ordered for execution; the only difference is that ordering within a batch is no longer considered unfair. The relaxation also is somewhat minimal in the sense that it is used on when the stronger notion is impossible. We provide more details in Section 3.4 and Appendix 3.9.

3.3 Fair Ordering Protocols

We describe the details for two fair ordering protocols, Π_{mod} and Π_{multi}, that each satisfy consistency, liveness, and batch-order-fairness in the permissionless setting.

Hammurabi design. To better motivate our design, we first take a step back to look at transaction ordering in standard Nakamoto consensus. Here, when the adversary mines a block, it has full control over the inclusion, exclusion, and ordering of transactions in the mempool. This essentially results in an *ephemeral* ordering centralization. To achieve fairness, we need to somehow decentralize the ordering process; this forms the basis for our design.

Our constructions follow the same outline. Through PoW mining, nodes mine d *semantic* chains, each of which represents a separate transaction ordering. We use the term *semantic* since the transaction ordering within different blocks can be logically linked even if the blocks themselves are not in the PoW sense. All transactions will be mined once into each of the semantic chains, with the overall

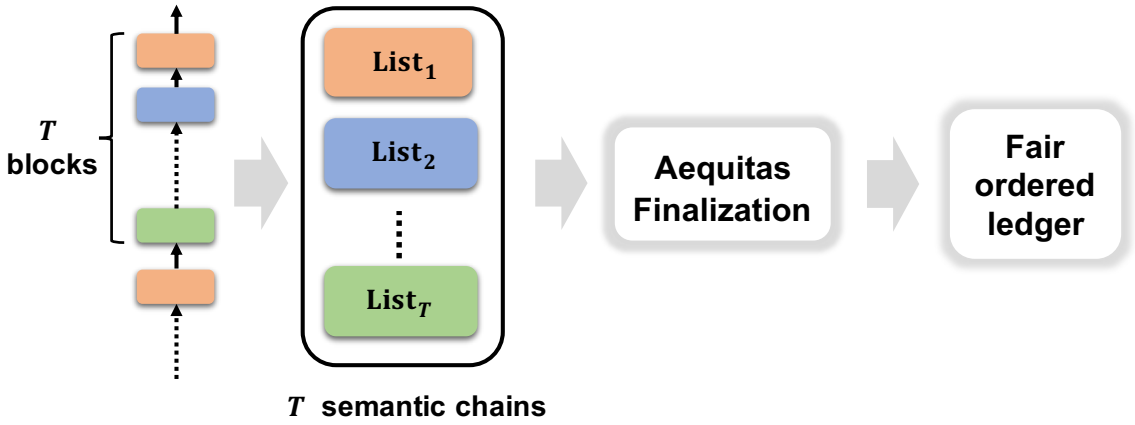


Figure 3-2: Pictorial representation of $\Pi_{\text{mod}}^{\kappa,T}$

fair ordering being extracted from the different orderings. The intuition is that an adversary will not be able to manipulate the ordering within a majority of chains. For our protocols, d is a function only of the security parameter κ and not of the number of nodes.

In the subsections that follow, we describe how our two protocols form these semantic chains. For now, we assume access to an algorithm $\mathsf{Aequitas}(\cdot)$ that extracts a fair final ordering from the semantic chains. Later, in Section 3.4, we distill the key structure of the protocol from KZGJ [104], and prove a novel player replaceability lemma which will allows us to implement $\mathsf{Aequitas}(\cdot)$ in the permissionless setting.

3.3.1 Modulo-T Longest Chain Protocol

We start with a basic order-fair protocol $\Pi_{\text{mod}}^{\kappa,T}$ which uses a single PoW-style longest chain, where κ is the security parameter, and T is the modular parameter. The protocol uses $d=T$ semantic chains but we use T here since it also corresponds to the consistency parameter of the underlying chain.

Protocol description. In $\Pi_{\text{mod}}^{\kappa,T}$, nodes mine a single PoW chain. For a given chain, $\mathsf{chain}[b]$ is considered to logically follow $\mathsf{chain}[b-T]$, despite its PoW hash following $\mathsf{chain}[b-1]$. In other words, the PoW chain will be split into T same semantic or "modular" chains, with blocks in the same index modulo T (e.g., indices 1, $T+1$, $2T+1$ and so on) describing a single transaction ordering. To obtain the fair ordering, $\mathsf{Aequitas}(\cdot)$ will be run on the semantic chains in the confirmed part of the PoW chain.

For a given chain, let $\mathcal{T}_b$ be the set of transactions that were first seen in block $\mathsf{chain}[b]$. Now, if $\mathsf{chain}[b]$ was honestly mined, then the first honestly mined block after $\mathsf{chain}[b]$ in each modular chain should contain these transactions. In other words, all honest nodes can reject blocks in the range $\mathsf{chain}[b+1:b+T-1]$ that do not contain all transactions in $\mathcal{T}_b$. Consequently, in a valid chain, the record $\mathtt{m}$ in any block $\mathsf{chain}[b]$ should be such that $\bigcup_{i=b-T+1}^{b-1}\mathcal{T}_i \subseteq \mathtt{m}$. Figure 3-3 contains a detailed description of Π_{mod} and Figure 3-2 contains a simplified pictorial representation.

Valid chain. A chain is valid iff (1) $\mathsf{chain}[0] = genesis = (0,0,\perp,0,H(0,0,\perp,0))$ is the genesis block; (2) $\mathsf{chain}[b].h_{-1} = \mathsf{chain}[b-1].h$ for all $b \in \{1,\cdots,|\mathsf{chain}|\}$; (3) $\mathsf{chain}[b].h' = genesis.h$ for $b \in \{1,...,T\}$ and $\mathsf{chain}[b-T].h$ for all $b \in \{T+1,\cdots,|\mathsf{chain}|\}$; and (4) $\bigcup_{i=b-T+1}^{b-1}\mathcal{T}_i \subseteq \mathsf{chain}[b].\mathtt{m}$ for $b \in \{1,\cdots,|\mathsf{chain}|\}$.

Modular chains. For a given valid chain, we say that two blocks $\mathsf{chain}[b]$ and $\mathsf{chain}[b']$ where b,b' are positive integers are in the same modulo-T chain if $b \equiv b' \mod T$. The genesis block, $\mathsf{chain}[0]$ is assumed to be included in all modular chains. We use $\mathsf{modularize}(\mathsf{chain},T)$ to denote the operation of splitting chain into modulo-T chains. Specifically, the output of $\mathsf{modularize}(\mathsf{chain},T)$ is $\mathcal{L} = [\mathsf{List}_1,...,\mathsf{List}_T]$ such

$\Pi^{\kappa,T}_{\text{mod}}(p)$

Initialize: $\mathsf{chain} = \textit{genesis}$; $\mathsf{TxPool} = \emptyset$

Upon receiving a valid chain' such that $|\mathsf{chain}'| > |\mathsf{chain}|$:

- Set $\mathsf{chain} := \mathsf{chain}'$ and output any newly seen transactions to $\mathcal{Z}$.

In round r, upon receiving txs from $\mathcal{Z}$:

- For each tx in txs, add (tx,r) to TxPool.
- Set h_{-1} to be the reference of $\mathsf{chain}[-1]$ and h' to be the reference of $\mathsf{chain}[-T]$ if $|\mathsf{chain}| > T$ otherwise set h' to be the reference of $\mathsf{chain}[0]$. Suppose that $\mathsf{chain}[-1]$ belongs to modular chain j.
- Let $\mathsf{txincludes}$ be the transactions in TxPool that are not any block in modular chain $(j+1 \bmod T)$, ordered by increasing round number.
- For $i = 1$ to q:
  - Pick a random nonce η and let $h = \mathsf{H}(h_{-1}, h', \mathsf{txincludes}, \eta)$.
  - If $h < D_p$, then set $\mathsf{chain} := \mathsf{chain} \,\|\, (h_{-1}, h', \mathsf{txincludes}, \eta, h)$, broadcast chain and break.
- Let $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_T] = \mathsf{modularize}(\mathsf{chain}[:-T], T)$
- Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as $\mathsf{linearize}(\mathcal{L})$.

$\mathsf{linearize}(\mathcal{L})$

- Output $\mathsf{Aequitas}(\mathcal{L})$.

Figure 3-3: Protocol $\Pi^{\kappa,T}_{\text{mod}}$

that List_j concatenates the transaction orderings in all $\mathsf{chain}[b]$ where $b \equiv j \bmod T$. We will use $\mathtt{ref} = h'$ to point to the previous block in the same modular chain.

Dynamic corruption. Recall that we restrict $\mathcal{A}$ to be (τ,R)-respawning, i.e., at most τn nodes can be killed within any R round period. Now, for $\Pi^{\kappa,T}_{\text{mod}}$, we will choose R such that, except with negligible probability, R is larger than the time it takes for a transaction tx to be confirmed in all modular chains. It turns out that choosing R to be larger than the time to mine $5T/2$ blocks suffices.

Compliant parameters. We define a compliance predicate for $\Pi_{\text{mod}}(p)$ as $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta,\gamma,T,\tau,R) = 1$ if n,Δ,T,R are polynomials in κ; the parameters β,γ,τ are constants such that $\gamma - (\beta+\tau) > \frac{2}{3}$; and for all κ, $\Gamma^{p}_{\text{nak}}(n,\beta,\Delta) = 1$, $\Pi_{\text{nak}}(p)$ satisfies T-common-prefix and $(T,\mu > \frac{1}{2})$-chain-quality, $R \geq \frac{5T+2}{2g_0}$ where g_0 is such that $\Pi_{\text{nak}}(p)$ also satisfies $(T,g_0,\cdot)$-chain-growth.

Theorem 5. *Consider parameters such that* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta = 1,\gamma,T,\tau,R) = 1$. *Then,* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *satisfies consistency, R-liveness, and* γ-batch-order-fairness *for* (n,β,Δ,τ,R) *environments.*

We provide an overview of the proof technique in Section 3.5 and defer the full proofs to Appendix 3.11.3. In Appendix 3.10.1, we also describe a variant that can tolerate any minority corruption using Fruitchains [133] as the underlying chain. We now state the main result for Π_{mod}.

3.3.2 Multi-Chain Protocol

We now introduce the protocol $\Pi^{\kappa,d}_{\text{multi}}$ (where d is the parallelization parameter), that uses multiple parallel Nakamoto chains.

Overview. Intuitively, in $\Pi^{\kappa,d}_{\text{multi}}$, honest nodes will mine d PoW chains in parallel, with each of the parallel chains serving as a semantic chain (i.e., an ordering for transactions). Similar to the single-chain protocol, $\mathsf{Aequitas}(\cdot)$ will now be run on the confirmed parts of each of the d chains. For simplicity, we will analyze the protocol only for $\gamma = 1$ and $\Delta = 1$. Following [134], we also use $p = \Theta(1/n)$ as the hardness parameter to make the per round mining rate $f = pqn$, be $\Theta(1)$ in κ. Figure 3-5 details our protocol and Figure 3-4 contains a simplified pictorial representation.

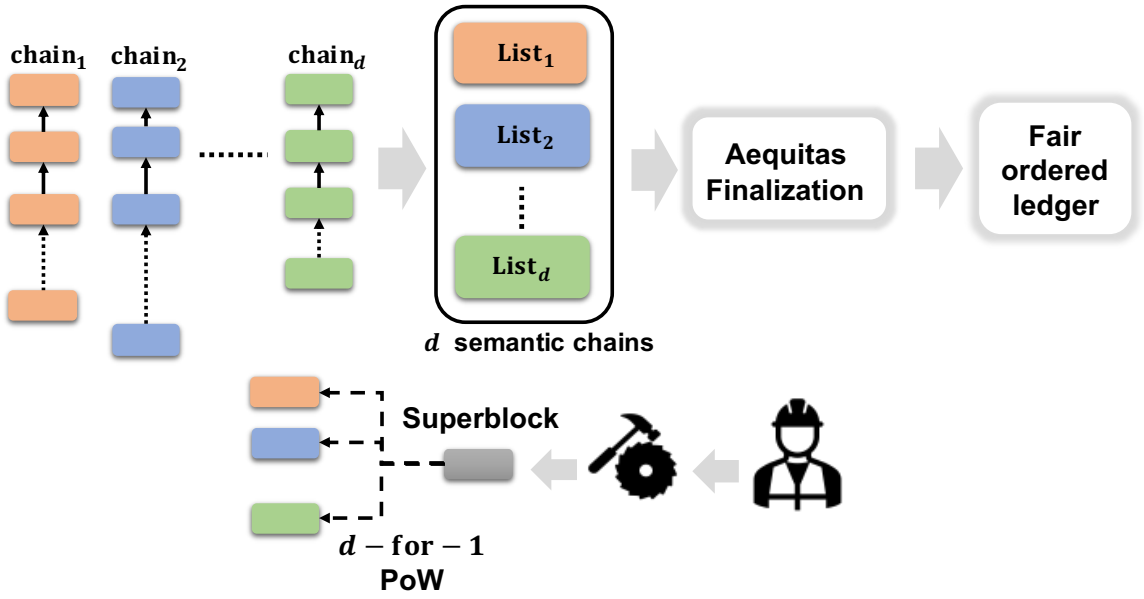


Figure 3-4: Pictorial representation of $\Pi^{\kappa,d}_{\text{multi}}$

d-for-1 PoW mining. To mine blocks, we use a d-for-1 mining technique, similar to the one from [28], which generalizes the 2-for-1 trick from [91, 133]. Abstractly, nodes will simultaneously mine transactions on all d chains, and the chain to which the block gets appended to will depend on the hash value.

For this, we will slightly abuse the notation for blocks from Section 3.2.2. Specifically, a block will be a tuple $(\mathbf{h}_{-1},\mathbf{h}',\mathbf{m},\eta,h)$ where $\mathbf{h}_{-1}$ will be a list of parent blocks on all d chains, $\mathbf{h}'$ will be a list of reference blocks on all d chains, and $\mathbf{m}$ is the list of transactions to be (potentially) appended to each chain. This is called a *superblock* in [28] but for simplicity, we use the same formalism of a block. For our purpose, we will have $\mathbf{h}_{-1}=\mathbf{h}'$. We will use a random oracle H that outputs $d\kappa$ bits (instead of κ). For hardness parameter $p(\kappa)$, define $D_p=p(\kappa)\cdot 2^{\kappa}$. For a block with hash $h=\mathsf{H}(\mathbf{h}_{-1},\mathbf{h}',\mathbf{m},\eta)$, we will say that it is "mined into chain i", if $h(i)<D_p$ where $h(i)$ is the substring of h in range $(i-1)\kappa$ to $i\kappa-1$. Considering different output bits for different chains allows us to make the mining for each chain independent although it does increase the size of the random oracle output. A block can be propagated to the rest of the network as part of the chain it gets mined in. For simplicity, we assume that the entire block is propagated, but note that in practice, similar to [28, 133], only the data relevant to its chain can be included, along with a commitment to the other discarded data.

Compliant parameters. We define a compliance predicate $\Gamma^{p}_{\text{multi}}(n,\beta,\Delta=1,\gamma=1,\tau,R,d,W)=1$ if n,R,W are polynomials in κ with $p=\Theta(1/n)$; β,τ are constants such that $\beta+\tau<\frac{7-\sqrt{17}}{16}$; and for all κ, $\Gamma^{p}_{\text{nak}}(n,\beta,\Delta)=1$, and there exists $T=T(\kappa)$ such that $\Pi_{\text{nak}}(p)$ satisfies T-common-prefix and $(T,\mu>\frac{1}{2})$-chain-quality, and $R(\kappa)\geq\frac{3T+2}{2g_0}$ where g_0 is such that $\Pi_{\text{nak}}(p)$ also satisfies $(T,g_0,\cdot)$-chain-growth. Furthermore, $W=\Theta(\kappa)$ and $d=\Theta(\kappa)$.

We will require R to be longer than the time it takes to mine $\frac{3T}{2}$ blocks in each chain. This will allow for tx to be mined and confirmed in all chains. W denotes the warmup time of our protocol.

We can now state the main result for Π_{multi}.

Theorem 6. *Consider parameters such that* $\Gamma^{p}_{\text{multi}}(n,\beta,\Delta=1,\gamma=1,\tau,R,d,W)=1$. *Then,* $\Pi^{\kappa,d}_{\text{multi}}$ *satisfies consistency,* (W,R)*-liveness, and* 1-batch-order-fairness *for* (n,β,Δ,τ,R) *environments.*

$\Pi^{\kappa,d}_{\text{multi}}(p)$

Initialize: $\mathcal{C} = [\mathsf{chain}_1, \ldots, \mathsf{chain}_d] = [genesis_1, \ldots, genesis_d]$; $\mathsf{TxPool} = \emptyset$; $T = T(\kappa)$ which is $\omega(\log\kappa)$

Upon receiving a valid $\mathcal{C}' = [\mathsf{chain}_1,\ldots,\mathsf{chain}_d]$:

- For all indices j, if $|\mathsf{chain}'_j| > |\mathsf{chain}_j|$, then set $\mathsf{chain}_j := \mathsf{chain}'_j$ and output any newly seen transactions to $\mathcal{Z}$.

In round r, upon receiving txs from $\mathcal{Z}$:

- For each tx in txs, add (tx,r) to TxPool.
- For all indices $j \in \{1,\ldots,d\}$:
  - Set both h^j_{-1} and h'_j to be the reference of $\mathsf{chain}_j[-1]$.
  - Let $\mathsf{txincludes}_j$ be the transactions in TxPool that are not any block in chain chain_j, ordered by increasing round number.
- Define $\mathbf{h}_{-1} = [h^1_{-1},\ldots,h^d_{-1}]$, $\mathbf{h}' = [h'_1,\ldots,h'_d]$, and $\mathbf{txincludes} = [\mathsf{txincludes}_1,\ldots, \mathsf{txincludes}_d]$.
- For $i = 1$ to q:
  - Pick a random nonce η and let $h = \mathsf{H}(\mathbf{h}_{-1},\mathbf{h}',\mathbf{txincludes},\eta)$.
  - If for some j, $h(j) < D_p$, then set $\mathsf{chain}_j := \mathsf{chain}_j \,\|\, (\mathbf{h}_{-1},\mathbf{h}',\mathbf{txincludes},\eta)$
  - If a nonce solution was found, then broadcast $\mathcal{C}$ and break.
- Let $\mathcal{L} = [\mathsf{List}_1,\ldots,\mathsf{List}_d] = \mathsf{extract}(\mathcal{C})$
- Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as $\mathsf{linearize}(\mathcal{L})$.

$\mathsf{extract}(\mathcal{C} = [\mathsf{chain}_1,\ldots,\mathsf{chain}_d])$

- Initialize all $\mathcal{L}_j$ to be empty.
- For all index $j \in \{1,\ldots,d\}$, and then for all $l < |\mathsf{chain}_j| - T$
  - Suppose that $\mathsf{chain}_j[l] = (\mathbf{h}_{-1},\mathbf{h}',\mathbf{txincludes},\eta,h)$. Append $\mathsf{txincludes}_j$ to $\mathcal{L}_j$.

$\mathsf{linearize}(\mathcal{L})$

- Output $\mathsf{Aequitas}(\mathcal{L})$.

Figure 3-5: Protocol $\Pi^{\kappa,d}_{\text{multi}}$

3.4 Extracting The Fair Ordering

We now detail how to extract a fair ordering from the semantic chains. To accomplish this, we first distill the key structure of the Aequitas protocol, as described in KZGJ , that provides order-fairness in the permissioned setting. We generalize their techniques to be useful in the permissionless setting we consider.

Background on Aequitas. Consider a permissioned system with n nodes, of which at most f are corrupt. We consider the synchronous Aequitas protocol which is parameterized by the order-fairness parameter $1/2 < \gamma < 1$ and works when $n > \frac{2f}{2\gamma-1}$. Roughly, Aequitas takes place in three stages that each transaction goes through. The first two stages involve each node gossiping the order in which they have received transactions and then agreeing on which nodes' local orderings to use to determine the final ordering. Intuitively, after this, all nodes have agreed on the same "data" used to order a given transaction. The third stage, dubbed the finalization stage, then, details a non-interactive algorithm used by nodes to extract the final ordering.

For our purpose, we abstract away the consensus components, i.e., the first two stages, and formulate the last stage as a *finalization* problem in Section 3.4.1.

3.4.1 The Finalization Problem

Before we formally define the finalization problem, we introduce some notation. For our finalization problem parameterized by $\Upsilon = (n,f,\gamma)$, we will consider as input, n ordered lists List_1 to List_n where an entry in a list is a set of messages. While the inputs to the finalization problem are independent of any underlying consensus problem, in the permissioned setting, intuitively, these lists describe the order in which messages were input to a particular node. We use $\mathsf{List}_i[r]$ to denote the the set of messages received by node i in round r (if i is honest). Lists corresponding to adversarial nodes, can of course deviate arbitrarily from their true input ordering. For messages m and m', we use the notation $m \prec_i m'$ if m is ordered before m' in List_i. The goal now, is to output a list $\mathsf{OutputList}$ of transactions that are ordered in a "fair" manner. We say that $\Upsilon = (n,f,\gamma)$ is admissible if $\frac{1}{2} < \gamma \leq 1$ and $n > \frac{2f}{2\gamma-1}$. We define a problem instance by $\Lambda = (\Upsilon, \mathcal{L})$ where $\Upsilon = (n,f,\gamma)$ and $\mathcal{L} = [\mathsf{List}_1,...,\mathsf{List}_n]$ indexed by 1 to n.

Definition 13 (γ-Global Preference)**.** *Given $\Lambda = (\Upsilon = (n,f,\gamma), \mathcal{L})$, for messages m and m', we say that m is globally preferred over m', denoted by $m' \lhd^{\Lambda} m$, if there are at least $\gamma n - f$ input lists List_j in $\mathcal{L}$ s.t. $m \prec_j m'$. We will write $m' \lhd m$ when Λ is clear from context.*

We define Condorcet cycles for non-transitivity in the global preference, which can arise from the Condorcet paradox (see Appendix 3.9 or [104]). Intuitively, this also makes receive-order-fairness impossible in the general case resulting in us using the batch relaxation.

Definition 14 (Condorcet Cycle)**.** *Given $\Lambda = (\Upsilon, \mathcal{L})$, we say that $(m_1, m_2, \ldots, m_l)$ is an l-length Condorcet cycle in $\mathcal{L}$ if $m_1 \lhd^{\Lambda} m_l$ and $m_{i+1} \lhd^{\Lambda} m_i$ for all $i \in \{1,...,l\}$.*

In Appendix 3.9, we show how a static version of the finalization problem has interesting connections to voting theory. Here, we directly formulate a novel streaming version of the problem, which better represents our setting.

Streaming finalization.

The primary difference that separates the Aequitas finalization protocol from prior work in voting theory is that the list of preferences is not fully determined at the start of the protocol. In the consensus setting, the protocol is ongoing as new transactions are continuously input to nodes[4]. Clearly nodes should not have to wait for all transactions to be input to decide on the fair ordering. Here, as noted by KZGJ , future transactions can cause changes in the condensation graph. Similarly, current transactions may not have been seen sufficiently many times so far. This means that current transactions might need to wait for future ones in order to be output in a fair order. The key technical challenge is to identify when transactions can be delivered without compromising on their fair ordering. We formalize this as the streaming finalization problem (Problem 1). Looking ahead, within our protocols, the d semantic chains will correspond to lists in this problem.

For lists List and List', we use $\mathsf{List} \preccurlyeq \mathsf{List}'$ to denote that List is a prefix of List'. For $\mathcal{L} = [\mathsf{List}_1,...,\mathsf{List}_k]$ and $\mathcal{L}' = [\mathsf{List}'_1,...,\mathsf{List}'_k]$, we also use the notation $\mathcal{L} \preccurlyeq \mathcal{L}'$ to denote that $\mathsf{List}_j \preccurlyeq \mathsf{List}'_j$ for all j.

[4]Intuitively, this is analogous to a growing list of potential candidates in an election, which is not particularly relevant for voting theory. See Appendix 3.9 for details.

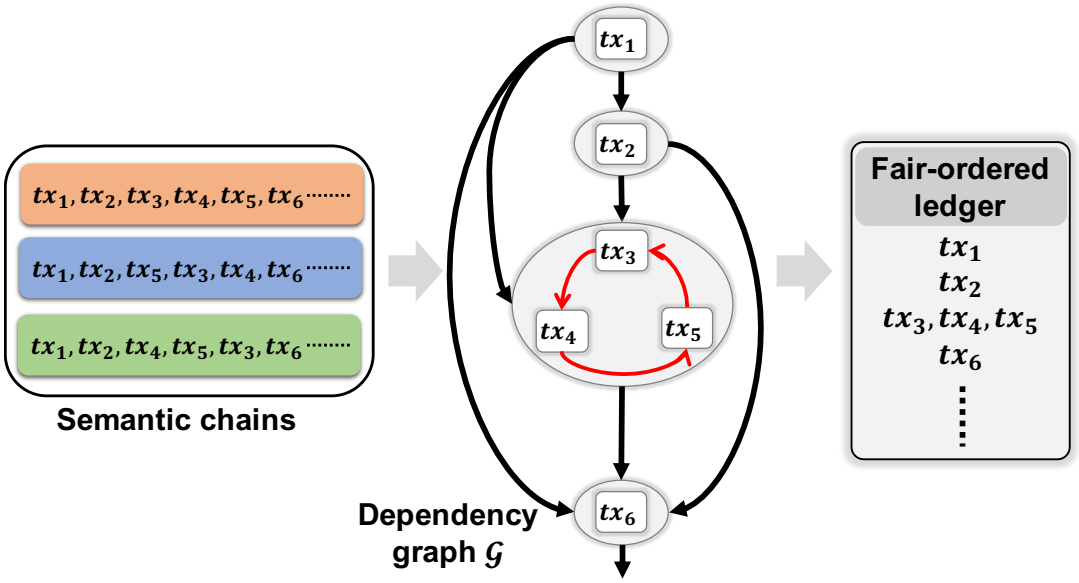


Figure 3-6: Simple Finalization Example

Problem 1 (Strong (resp. Weak) Streaming Finalization)**.** *Consider an admissible* $\Upsilon = (n, f, \gamma)$*. At each timestep* r*, consider as input* $\mathcal{L}^r = [\mathsf{List}_1^r, \ldots, \mathsf{List}_n^r]$ *such that* $\mathcal{L}^{r_1} \preccurlyeq \mathcal{L}^{r_2}$ *whenever* $r_1 \leq r_2$*. At timestep* r*, output a list* $\mathsf{OutputList}^r$ *such that:*

- *(Prefix-Consistency)* $\mathsf{OutputList}^{r_1} \preccurlyeq \mathsf{OutputList}^{r_2}$ *whenever* $r_1 \leq r_2$*.*
- *(Strong (resp. Weak) Fairness of ordering) For all* (m, m') *such that* $m' \lhd^{\Lambda} m$*, where* $\Lambda = (\Upsilon, \mathcal{L}^r)$*, if* $\mathsf{OutputList}^r$ *contains* m *or* m'*, then* m *is ordered before (resp. no later than)* m'*.*

An algorithm $F(\cdot)$ solves the streaming finalization problem for an admissible Υ if for any valid input sequence $\mathcal{L}^1, \mathcal{L}^2, \ldots$, for every timestep r, $F(\mathcal{L}^r)$ satisfies prefix-consistency and fairness of ordering. Note that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $m' \lhd^{(\Upsilon, \mathcal{L})} m$ implies $m' \lhd^{(\Upsilon, \mathcal{L}')} m$ since for any $\mathsf{List}_j \in \mathcal{L}$ where $m \prec m'$, the corresponding $\mathsf{List}'_j \in \mathcal{L}'$ will also have $m \prec m'$. In other words, the fairness of ordering condition will not contradict the prefix-consistency condition. Note that liveness is not included in Problem 1 (analogous to how delivering no transactions still satisfies "consistency" in the consensus abstraction). Liveness will be a separate protocol requirement.

Aequitas finalization [104]. Given the n input orderings from the protocol nodes, the Aequitas finalization stage builds a dependency graph of transactions. An edge (m, m') is added whenever m is globally preferred over m'. Transactions can be output, when intuitively, they can no longer have incoming edges from a different strongly-connected-component (see Appendix 3.9). KZGJ showed that whenever m is globally preferred to m', the Aequitas finalization will order m no later than m' in the general case, and m strictly before m' when in a setting where there are no Condorcet cycles. In turn, this directly implies that the Aequitas finalization solves our (weak) streaming finalization abstraction where the input lists for honest nodes are the actual ordering of inputs from $\mathcal{Z}$. For completeness, this is also shown in Appendix 3.11.1. Our streaming formalism primarily enables us to move to the permissionless setting through player replaceability.

For our purpose, we can abstract out the underlying protocol nodes and run the Aequitas finalization only on the lists. In general, given $\Upsilon = (n, f, \gamma)$ and $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_n]$, we will use $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$ to denote the dependency graph and $\mathsf{Aequitas}_{\Upsilon}(\mathcal{L})$ to denote the resultant outputs of applying the finalization step of the Aequitas protocol on the lists in $\mathcal{L}$. We drop the subscript when Υ is clear from context. Note that in the consensus context, since different nodes may construct the semantic chains

at potentially different times, the actual dependency graph of a node will be based on the parts of the semantic chains the node has confirmed, and therefore different nodes may have slightly different graphs. We provide a simple example what finalization looks like in Figure 3-6. Cycles represent non-transitive preferences that will be output in the same "batch" for batch-order-fairness . Recall that here, transactions within a batch can still be totally ordered for execution; the only difference is their ordering is no longer considered unfair.

Player Replaceability

So far, the input lists used for $\mathsf{Aequitas}(\mathcal{L})$ were the actual transaction input orderings (for honest nodes) in a permissioned protocol. The key idea we use to move to the permissionless setting is to enable nodes to "continue" the orderings of other nodes in the system. In particular, we will fix a number of input lists, say d, at the start of the protocol. All nodes now extend the existing transaction orderings in these d lists. This can be done through any permissionless consensus technique (e.g., PoW, PoS etc). The d lists can be thought of as the input orderings of d "virtual" nodes in a permissioned network. The Aequitas finalization algorithm will now be run on the d lists to determine the final output ordering. To allow for a seamless reduction of the permissionless problem to a permissioned one, we use a new *player replaceability* lemma. Intuitively, this lemma will directly let us conclude order-fairness for our permissionless protocols, by proving order-fairness for the permissioned transformation with d virtual parties.

Transformation property. The ability to use the "virtual" party transformation can be formalized as a completeness property of the dependency graph. Informally, there should be an edge from tx to tx$'$ in the dependency graph of any honest node whenever (tx,tx$'$) satisfies the antecedent of the order-fairness definition. Note that we consider the graph built from the full output $\mathcal{L}$ of the node and not the graph that it retains after delivering other transactions. This allows us to discuss an edge between transactions even though one of them might have been delivered in the final ledger.

Property 1 (Completeness of Dependencies)**.** *Consider* (n,β,Δ,τ,R) *and an order-fairness parameter* γ*. We say that* Π *is* γ*-dependency-complete if for any* $(\mathcal{A},\mathcal{Z})$ *that satisfies* (n,β,Δ,τ,R)*-permissionless execution, the following holds: If* (tx,tx$'$) *satisfies the antecedent of the* γ-batch-order-fairness *definition, then for honest node* N *in any round* r*, if* N *outputs* $\mathcal{L}$ *where* tx$'$ *is in all lists in* $\mathcal{L}$*, then* $\text{tx}' \lhd^{\Lambda} \text{tx}$ *holds where* $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$*. In other words,* $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$ *contains the edge* (tx,tx$'$)*.*

We can now describe the player replaceability lemma. We provide a simpler version that suffices for our purpose here, but prove a more general result in Appendix 3.11.1.

Lemma 17 (Player Replaceability)**.** *Consider an* $(\mathcal{A},\mathcal{Z})$ *that satisfies* $(n,\beta,\Delta,\cdot,\cdot)$*-permissionless execution and an order-fairness parameter* γ*. Suppose that a protocol* Π *is* γ*-dependency-complete and satisfies the following consistency-like property (from Problem 1):*

- *For any node* N *that is honest in a round* r*,* N *outputs* $\mathcal{L}_N^r = [\mathsf{List}_1^r, \ldots, \mathsf{List}_d^r]$ *to* $\mathcal{Z}$*.*
- *If* N *is honest in any rounds* $r-1$ *and* r*, then* $\mathcal{L}_N^{r-1} \preccurlyeq \mathcal{L}_N^r$*.*

Suppose that an algorithm $F(\cdot)$ *satisfies the strong (resp. weak) streaming finalization problem for* Υ *and that the* linearize *function of* Π *is defined to be* $F(\mathcal{L}_N^r)$ *for node* N *and round* r*. Then,* Π *satisfies* γ*-receive-order-fairness (resp.* γ*-batch-order-fairness) w.r.t.* $(\mathcal{A},\mathcal{Z})$*.*

Intuitively, as long as a protocol Π that satisfies the specified properties (i.e., it provides a transformation to d virtual parties), the upshot of this lemma is that now, we can run $F(\cdot) = \mathsf{Aequitas}(\cdot)$ on the d semantic chains to determine the final fair output ordering. In Section 3.5, we show why our protocols confer to the prerequisite of Lemma 20 thereby achieving batch-order-fairness .

3.5 Proofs

3.5.1 Overview of Proof Technique

We overview our key techniques for proving order-fairness and liveness for our protocols Π_{mod} and Π_{multi}. We note that the consistency of our protocols will directly follow from the consistency of the underlying Nakamoto chain.

Order-Fairness. Recall that in our protocols, honest nodes output the semantic chains to $\mathcal{Z}$ in an append-only fashion, and that the $\mathsf{linearize}(\cdot)$ function (which decides the final ledger) is defined as $\mathsf{Aequitas}(\cdot)$ (which solves the (weak) finalization problem) on the semantic chains. Therefore, it suffices to show dependency-completeness in order to use Lemma 20 and conclude batch-order-fairness . In Appendix 3.9, we briefly remark on the potential to achieve receive-order-fairness in the permissionless setting.

Liveness. For liveness, we start with the following faithfulness property. Informally, a protocol is *dependency-faithful* if some honest node has received tx before tx′ whenever an edge from tx to tx′ exists in the dependency graph of an honest node.

Property 2 (Faithfulness of Dependencies)**.** *Consider* (n,β,Δ,τ,R)*. We say that* Π *is dependency-faithful if for any* $(\mathcal{A},\mathcal{Z})$ *that satisfies* (n,β,Δ,τ,R)*-permissionless execution, the following holds: For honest node* N *in any round* r*, if* N *outputs* $\mathcal{L}$ *with* $\text{tx}' \lhd^{\Lambda} \text{tx}$ *where* $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$*, then there is at least one honest node that has received* tx *before* tx′ *as input from* $\mathcal{Z}$*. In other words, if an edge* (tx,tx′) *exists in* $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$*, then some honest node must have received* tx *before* tx′*.*

Now, through Lemma 18 (full proof in Appendix 3.11.2), we show that if a protocol is dependency-faithful, then when $\Delta = 1$, if transactions tx and tx′ (even adversarial) are in the same Condorcet cycle, then they cannot have been received far apart. In other words, a transaction does not have to wait for an arbitrary length of time to ensure that it does not get combined with a later transaction in the same strongly connected component of the dependency graph. Consequently, this will enable transactions to be delivered in a bounded length of time satisfying liveness. Note that the specific liveness bound will be dependent on the actual protocol.

Lemma 18. *Consider* $(n,\beta,\Delta = 1,\tau,R)$ *and suppose that a protocol* Π *is dependency-faithful (Property 2). Suppose that some honest node in round* r *outputs* $\mathcal{L}$ *where* tx *and* tx′ *are in the same Condorcet cycle. Let* r,r' *be the first round that some honest node has received* tx *and* tx′ *respectively. Then* $r = r'$*.*

To summarize, it will be sufficient to show that are protocols are both dependency-completeness and dependency-liveness.

3.5.2 Proofs for Π_{mod}

We first note that consistency of $\Pi_{\text{mod}}^{\kappa,T}$ comes for free (when $\beta < \frac{1}{2}$) from the consistency property of Nakamoto's blockchain since we use a single PoW chain. For batch-order-fairness and liveness, we show that $\Pi_{\text{mod}}^{\kappa,T}$ is both dependency-complete and dependency-faithful. This is a relatively straightforward consequence of the chain-quality of the underlying Π_{nak}. Here, we sketch the proof for $\gamma = 1$, and defer the general result and full proof to Appendix 3.11.3.

Completeness proof sketch. Consider (tx,tx′) such that all honest nodes received tx before tx′. Let $\mathsf{chain}[b]$ be the first block that contains either tx or tx′. Note that all blocks in $\mathsf{chain}[b\!:\!b+T-1]$ will also contain at least one of tx and tx′, thereby fixing the ordering between them in all semantic chains. Now, by chain-quality of the underlying Π_{nak}, more than half of the the blocks in $\mathsf{chain}[b\!:\!b+T-1]$ are mined by honest nodes, i.e., will include tx earlier. Consequently, any honest dependency graph will contain the edge (tx,tx′).

Faithfulness proof sketch. Suppose that an honest dependency graph contains the edge (tx,tx′) (i.e., $\mathrm{tx}' \lhd \mathrm{tx}$ holds). Let $\mathsf{chain}[b]$ be the first block that contains either tx or tx′; the blocks in $\mathsf{chain}[b : b + T - 1]$ then order the transactions in all semantic chains. Now, more than half of the semantic chains contain tx earlier but by chain-quality of the underlying Π_{nak}, less than half of them can be adversarial. Therefore, at least one honest node has mined a block with tx ordered earlier.

3.5.3 Proofs for Π_{multi}

Once again, consistency comes for free (when $\beta < \frac{1}{2}$) from the consistency of the underlying PoW chains. To prove order-fairness and liveness, we first define a cross-chain-quality as Property 3, and show that it implies both dependency-completeness and dependency-faithfulness for Π_{multi}. Finally, we show that Π_{multi} satisfies cross-chain-quality. The proofs for Π_{multi} are arguably more involved than those for Π_{mod} and we provide only a brief sketch here. The full proofs are deferred to Appendices 3.11.4 and 3.12.

Property 3 (Cross-Chain-Quality)**.** *For a given* view *and* tx*, let* $\mathcal{B}_{\mathrm{tx}}$ *be the set of blocks that contain* tx *for the first time in the confirmed part (i.e.* $\mathsf{chain}_j[:-T]$*) of each of the* d *chains. Then, more than half of the blocks in* $\mathcal{B}_{\mathrm{tx}}$ *were mined by honest nodes.*

Cross-chain-quality implies dependency-completeness and dependency-faithfulness (Lemma 24). We prove this by contradiction. For any tx and tx′, we consider all the possibilities for how the transactions are mined in a given chain (e.g., by an honest or adversarial node) and how they were ordered in that chain (e.g., tx before tx′, tx after tx′, or at the same time). Then, we use a simple combinatorial argument to show that if dependency-completeness or dependency-faithfulness were not satisfied, then either tx or tx′ would be need to be mined by the adversary in at least half of the parallel chains which contradicts cross-chain-quality.

Intuitively, this proof can be thought of as a generalization of the idea used to prove completeness and faithfulness of Π_{mod}; specifically, cross-chain-quality now plays an analogous role to the chain-quality property of Π_{nak} used earlier in the proofs for Π_{mod}.

Cross-chain-quality proof (Appendix 3.12) The bulk of the technical novelty lies in showing that Π_{multi} satisfies the cross-chain-quality property. Roughly, we employ the following strategy. First, for a chain i and round r, we construct an event $V_i[r]$ such that whenever $V_i[r]$ holds, any transaction that is first seen by honest nodes at round r will be mined into an honest block. Next, we show that for a sufficiently small hardness parameter, the probability that this event happens, i.e., the transaction is mined into an honest block is more than 0.5. This means that by setting the number of chains to be large enough, we can now use the Chernoff bound to conclude that the transaction is contained in honestly mined blocks in more than half the chains, except with negligible probability. Equivalently, the cross-chain-quality property will be satisfied, except with negligible probability. Our proof makes novel use of techniques from the queuing theory literature and may be of independent interest. We point the reader to [79, 96] for a useful primer on the techniques we use.

To bound the probability for the event $V_i[r]$, we break it into two independent events $V_i^1[r]$ and $V_i^2[r]$ as follows:

- $V_i^1[r]$ is the event that in round r, there is no private adversarial chain that is longer than the chain i.
- $V_i^2[r]$ is the event that the first block mined in chain i after round r is honest and the block persists in the chain forever (except with negligible probability).

At a high level, under the event $V_i[r]$, if a transaction is first seen in round r, then it will be contained in the first honest block mined after round r in chain i and it will persist. Now, we use techniques from queuing theory to compute the probabilities for each event. We highlight the non-triviality of computing the probabilities—primarily due to the possibility of adversarially propagated transactions. These are transactions that the adversary generates but does not immediately broadcast, instead attempting to first mine them into several parallel PoW chains. Abstractly, we need to carefully consider the

scenario where an adversary privately mines a transaction onto several chains before releasing all the mined blocks at once, thus giving it a head start before honest nodes even see this transaction. We need to prove that even with such a head start, in more that half of the parallel chains, the transaction will be contained in an honestly mined block.

To compute the probability for $V_i^1[r]$, we first formulate the event as a discrete-time Markov chain (DTMC) and then show it to be positive recurrent and irreducible. We do this in order to use the ergodic theorem [96] and relate $P(V_i^1[r])$ to the stationary distribution of the DTMC. Finally, we use z-transforms to compute the stationary distribution probability. For $V_i^2[r]$, we first model the event as a one-dimensional random-walk which counts the difference between the number of honest blocks and the number of adversarial blocks mined in a given period. $P(V_i^2[r])$ can now be formulated as the probability that the random walk starting from an initial positive state never goes to zero (after a small warmup period). We use z-transforms to compute the random-walk probabilities.

3.6 Applications

3.6.1 Zero-Block Confirmation

We show how fair ordering protocols can confirm non-conflicting transactions without mining a single block in the synchronous setting. We start by defining a *soft-ordering* property.

Definition 15 (δ-Soft-Ordering)**.** *A protocol* Π *satisfies* δ*-soft-confirmation if the following property holds: If, an honest node receives* tx *in round* r *and* tx′ *in a round greater than* $r+\delta$*, then the final log contains* tx *before* tx′*.*

Now, define conflict(tx,tx′) = conflict(tx′,tx) = 1 if tx and tx′ conflict with each other according to some semantic (e.g., spend the same tokens in a UTXO system). If two conflicting transactions are present in the output chain, only the first one will be executed by honest nodes to determine the current system state. Suppose that a node N receives tx in round r and is honest till at least round $r+\delta$. If N does not receive any tx′ such that conflict(tx,tx′)=1 till round $r+\delta$, then all honest nodes will output tx before any conflicting tx′, and thus tx will be confirmed in the finalized ledger (due to the δ-soft-ordering property). This gives rise to a primitive that can be used to soft-confirm transactions. If δ is small, no blocks containing tx will be mined by this time, yet N will be able to soft-confirm tx, i.e., we get zero-block confirmation. Finally, we note that a transaction submitted by an honest node will not have any conflicting transactions at a future time, and thus can be confirmed by any other honest node within δ rounds.

It is easy to see that receive-order-fair protocols will satisfy 2Δ-soft-ordering since transactions received more than 2Δ apart will always be such that all nodes receive the first transaction before the second. Consequently, they also provide zero-block-confirmation. Note that the probability that a soft-confirmed transaction is not valid in the finalized ledger is the same as the probability that the protocol does not satisfy order-fairness, which is negligible in κ.

Theorem 7. *If* Π *satisfies* γ*-receive-order-fairness w.r.t.* $(\mathcal{A},\mathcal{Z})$ *with network parameter* Δ*, then it also satisfies* (2Δ)*-soft-ordering.*

Remark 3. *Although our protocols satisfy* batch-order-fairness *, and not receive-order-fairness, when* $\Delta = 1$*, they will satisfy* (2Δ)*-soft-ordering since any Condorcet cycle cannot extend past this time (Lemma 18), and therefore we still get zero-block-confirmation.*

Soft ordering as Hamiltonian paths. While the previous application of soft-ordering allows fast confirmation of non-conflicting transactions, we provide another angle in terms of the possible orderings the final log could contain. For this, we relate possible final orderings to Hamiltonian paths in a graph built only using a node's local ordering. We defer the details to Appendix 3.11.6.

Network considerations. While soft-ordering is an easy corollary of order-fairness, we emphasize that is not a direct artifact of network synchrony in a permissionless setting. Even when transactions are gossiped and reach all nodes within a synchronous Δ period, soft-confirmation within a small delay cannot be done if the underlying consensus protocol is not fairly ordered. For instance, suppose that a node N received tx in round r and a conflicting tx$'$ in round $r+2\Delta+1$, i.e. all nodes receive tx before tx$'$. Still N cannot soft-confirm tx, since a miner proposing a block with tx$'$ first may not be rejected by the network (since in the view of a honest node who receives tx in round $r+\Delta$ and tx$'$ in round $r+\Delta+1$, it is plausible that other nodes have received tx$'$ first.)

Larger soft-confirm values (small enough to still be zero-block) do not work either. Even if it is common knowledge for the current nodes that everyone received tx before tx$'$, the rejection of an adversarial miner's block with tx$'$ first, will require this additional information. This means that the determination of whether a chain is valid is no longer only dependent on the transactions in it. Consequently, when new nodes enter the system, they will not be able to determine the correct honest chain to mine on.

Another concern is nodes not gossiping double-spends as a mechanism to prevent DoS attacks. Fortunately, gossiping only the first instance of double-spend for a token is sufficient for an honest node to know not to locally soft-confirm a transaction; other double-spends for the same token need not be gossiped to the entire network. This allows us to retain our zero-block-confirmation property will still ensuring that flooding the network will not cause a DoS attack. It also does not affect honest transactions which are not double-spent.

Decentralized finance. The use of fair ordering protocols can greatly benefit decentralized finance applications. Consider automated market makers (AMMs) like Uniswap [14], Curve [9], etc., that allow users to exchange between tokens using an in-built price function to calculate the exchange rate. Here, the token exchange price that a user gets depends on the time her transaction executes. Using a fair ordering protocol can guarantee that users receive a fair price for their trades. Furthermore, it will also prevent miners from inserting their own transactions to take advantage of short-term changes in token exchange rates.

Order-fairness also grants fairness to applications whose incentives are based on earlier ordering. For example, in sealed-bid auctions, bids submitted before the auction close time cannot be forcefully rejected by a miner claiming to have received them later. Similarly, in an initial coin offering (ICO) token launch, if the developer wants to provide a cheaper rate for the first 1000 tokens, it will be able to do so in a fair "first come first serve" manner.

3.7 Related Work

We describe related works in this section. The SMR abstraction has been studied in literature for several decades but almost no prior permissioned protocols, and no permissionless protocols considers any notion of fairness in transaction ordering.

Fair ordering protocols. Very little research has been done on fair ordering consensus protocols. Hashgraph [29] provided an informal description of a first-in-first-out (FiFo) transaction ordering property. KZGJ [104] formally defined (receive)-order-fairness (resp. batch-order-fairness) as the property of ordering tx_1 before (resp. no later than) tx_2 globally if a large fraction of nodes (parameterized by γ) received tx_1 before tx_2 in their local ordering. KZGJ provided permissioned protocols that achieved their definitions. Concurrently to KZGJ , works by Kursawe [110], and Zhang et al. [167] gave an alternative definition for ordering fairness which we compare below. Here, the fair ordering problem is tackled at a *pre-protocol* stage, i.e., the ordering is determined independently and before the actual consensus protocol is run.

Comparison to other fairness definitions. Kursawe [110], and Zhang et al. [167] independently defined a different notion of fair ordering (called *timed relative fairness* in [110] and *fair-linearizability* in [167]) in the permissioned setting where tx_1 is ordered before tx_2 if all honest nodes have seen tx_1 before *any* honest node has seen tx_2. A similar property was also stated as a desideratum in [87] although

no protocols achieving this property were given. Note that this is strictly weaker than the receive-order-fairness property from KZGJ and in Section 3.2.3. Further we find that its antecedent can easily made false, and therefore it is not enough to prevent natural order-manipulation attacks. This is because it enforces no guarantees on transactions whose receive times are globally interleaved. An adversarial transaction that attempts to front-run an honest user transaction, will end up getting interleaved with the user transaction in its receive times. Consequently, for most practical attack vectors, the antecedent of fair-linearizability will be false, making the definition vacuous and not particularly useful. We also note that this definition lacks a parameterization by γ, compared to the order-fairness definition.

Another natural definition is based on median timestamps; specifically, tx_1 is ordered before tx_2 if the median receive time for tx_1 is smaller than that for tx_2. However, KZGJ showed that that this does not model a FIFO notion as a single adversarial node can cause tx_2 to be ordered earlier even when all nodes have received tx_1 earlier.

We therefore chose to extend the order-fairness definition from KZGJ . We emphasize however, that our techniques are general enough to be adapted to provide other notions of fair ordering.

Alternate ordering techniques. We compare the general idea of fair ordering protocols with several prior techniques used to constrain adversarial influence over transaction ordering.

Censorship Resistance. An extensive line of research ([91, 132, 133] among others) in the past few years has considered the consensus problem in the permissionless setting. By design, most permissionless protocols achieve *censorship resistance*, which ensures that any transaction submitted by an honest user will eventually be confirmed by the network. Despite this, no guarantees are provided on how the transaction is eventually ordered.

Content Hiding. Some protocols make the use of threshold encryption [41, 143, 23, 126], or a commit-and-reveal scheme [37, 22], to order transactions before their content is revealed in an attempt to prevent adversarial ordering. In practice, this might still not prevent censorship and reordering based on transaction metadata (user identifier, client IP address etc) or front-running based on information received through side channels from colluding clients [104]. Furthermore, an increasing number of attacks (e.g., displacement [75] and certain backrunning [122]) do not depend on transaction contents and therefore will not be stopped by these techniques.

Random Ordering. It is important to distinguish a protocol that chooses a *random* ordering from a protocol that chooses a *fair* ordering. A *random* ordering protocol chooses a random ordering of transactions in the current pool, in an attempt to avoid any adversarial manipulation. On the other hand, a fair ordering protocol will guarantee that transactions which arrive earlier will be sequenced earlier by the network (instead of in a random order). Even in a hypothetical perfect random ordering protocol where no metadata or side-channel information is leaked and where adversarial miners are provably forced to choose a random ordering of transactions in the mempool, there is still a 50% chance that an adversarial transaction gets ordered before an honest one even if the honest one was propagated earlier. In addition, the adversary can flood the network with many transactions to exponentially increase the probability that at least one is ordered before the honest transaction (e.g., clogging attacks [139]). We highlight that even if the adversary has no control over the ordering of transactions in the pool, it is able to implicitly influence the ordering by introducing its own adversarial transactions.

Random ordering is especially problematic if the adversary can use user transaction metadata to construct its own transaction, or if the adversarial transaction does not need to depend on the contents of any honest user's transaction (e.g., displacement attacks [75] where the adversary wants to be the first one through the door to for instance, buy an asset with only limited quantities available, or get the best price in an auction). This is not a hypothetical problem as such a flooding attack was performed during the BZRX token launch [122] to guarantee a better token buy price for the adversary. Therefore, a random ordering protocol is not sufficient to prevent order-manipulation attacks. In contrast, our fair ordering protocols ensure that an adversary cannot tip the scales in its favor by creating many transactions.

Other unrelated uses of fairness. The term *fairness* has been used previously in literature for properties unrelated to ours. In permissionless consensus literature, one common use case is for PoW block mining; it requires that an honest node's rewards be proportional to its relative computational

power, which restricts adversarial *selfish mining* [77]. This notion of fairness was first defined by Pass and Shi [133] and achieved by their Fruitchains protocol. We will call this property *mining-fairness* to distinguish it from our notion of order-fairness. A similar notion is also considered in the permissioned setting [113, 126] where fairness is defined in terms of the opportunities each participant gets to append transactions to the ledger. Abstractly, this is equivalent to choosing a random node (or committee) to append the next block to the ledger. We highlight that this still does not provide any form of fair transaction ordering since malicious nodes can still reorder transactions at will in the blocks that they propose. Fairness has also been used in the context of *robustness*—guaranteed output—for example in [22, 116].

Latency reduction methods. Reducing the confirmation latency of longest-chain protocols has been the subject of intensive recent research [82, 166, 114, 28]. Two particularly relevant works, Prism [28] and Ledger-Combiner [82], also use multiple parallel chains to achieve faster confirmation. They can confirm honest transactions within a constant multiple of the network delay Δ, much faster than Bitcoin, where the latency is a multiple of the security parameter. However, in both methods, the constants are large and become worse as the adversarial power increases. Other methods [114, 137] offer low latency only under optimistic conditions, and fall back to standard Bitcoin latency when there are any adversarial actions. In comparison, our soft-confirmation method can confirm honest transactions in 2Δ rounds independent of the tolerable adversary ratio or whether there is an active attack. Conceptually, our soft-confirmation method can confirm transactions that are yet to get into any block, which is impossible with the other methods. Surprisingly, Ledger-Combiner, which was specifically designed to get low latency when combining transactions from multiple ledgers is slower than the soft-confirmation our constructions achieve as a byproduct. Recent work in distributed systems [31] has shown how to utilize "non-skipping time-stamps" to locally commit transactions as soon as nodes know that no conflicting transaction can be ordered earlier, and termed this property *clairvoyance*, emphasizing the look-ahead ability. [31] proposes protocols that achieve clairvoyance, but are specific to the permissioned setting, and even there, are not designed for, and hence do not achieve, our stronger definition of fair ordering.

3.8 Model (cont.)

3.8.1 Network Model Intuition

We delve into the subtleties of our network modeling that were deferred from the main body. Standard formalism for consensus protocols models a single communication network amongst consensus nodes. Informally, network synchrony here implies that any message sent by an honest node reaches its intended recipient(s) within some known fixed amount of time (or rounds) Δ. Usually transactions from clients are assumed to originate either from a consensus node or input by the environment $\mathcal{Z}$.

The order-fairness formalism from KZGJ , however, models two separate networks: an external network (which models the communication channel between system users and protocol nodes), and the internal network (which models the communication network amongst protocol nodes and is the standard one considered in literature). Furthermore, the model assumes that transactions are sent by clients to all nodes (modeled as inputs given by $\mathcal{Z}$), and that the adversary only controls network delivery for the internal network. Synchrony in the external network is defined as the property that any transaction input to one node (by $\mathcal{Z}$) is input to all other modes within some known time Δ_{ext}. In practice, this is the assumption that a client's connection to any particular node is not much slower than its connection to others.

Such a modeling was crucial for their purpose because each consensus node needed to have an unmolested view of its transaction input ordering. For example, the first node that receives a client transaction should not be able to adversarially insert its own transaction and quickly gossip it to other nodes in an attempt to get it sequenced first. This is a form of "front-running" as described in [63]. The assumption that the adversary does not control the external network takes care of this problem. However, this puts the onus on the client to send its transaction to all nodes instead of just a few. While this is a perfectly reasonable assumption in a permissioned network where the nodes are known, it is

difficult to justify it in a permissionless network, where nodes change over time and transactions usually propagate through a gossip-style network. Here, arguably there is no way to get around nodes inserting their own transactions during the gossip. Nevertheless, we emphasize that this still retains the notion that a transaction that is seen by a majority of the network first, should be sequenced first. We note that the permissioned formalism in KZGJ could very well have modeled a single network, but the additional modeling of the external network allowed an honest node's input ordering to be completely independent of the adversary. In practice, in a permissionless network, clients should send their transactions to as many nodes as they can so that most nodes receive them in an order not influenced by the adversary.

Another upshot of modeling two separate networks in KZGJ was that a transaction claimed by an adversarial node would not be considered by an honest node unless the transaction was received from the environment i.e. an honest client. This works well since now, an adversary cannot influence the global state by creating bogus transactions. Unfortunately, in a permissionless Proof-of-Work protocol, transactions can also be gossiped by mining a block that contains it and propagating the block through the network. It is difficult to determine the final ordering simply based on the local input orderings of all the nodes from clients (like was done in KZGJ) since participating nodes may constantly change in a permissionless system. On the other hand, if transaction orderings are based on the ordering in the ledger, it could potentially enable an adversarial transaction (that was not input to any honest node), to be sequenced earlier simply by being placed earlier in the ledger. To get around this, informally, we force the environment to deliver transactions included in the ledger to all nodes *as input* which allows us to talk meaningfully about transaction ordering. This is captured in the ledger synchrony part of the definition

All of this ends up blurring the distinction between the internal and external network from KZGJ , leading us to model a single communication network.

3.8.2 Standard Blockchain Formalism

For completeness, we include the standard abstract formalism for blockchain protocols. An abstract blockchain chain is an ordered sequence of blocks where $\mathsf{chain}[i]$ is a vector of transactions contained in chain at index i. We define the requirements for an abstract blockchain next.

Definition 16 (Abstract Blockchain Requirements)**.** *Except with negligible probability over a random choice of* view *in the support of* $\mathsf{EXEC}^{\Pi}(\mathcal{A},\mathcal{Z},\kappa)$*, the following need to hold:*

- *(T-Common-Prefix) If* chain^r *and* chain^t *are two honest chains (possibly from the same node) in rounds r and t respectively, and $r \leq t$, then* $\mathsf{chain}^r[:-T] \preceq \mathsf{chain}^t$*. This will allow us to confirm all except the last T blocks in* chain*.*
- *((T,g_0,g_1)-Chain-Growth) Let* chain^r *and* chain^t *be two honest chains (possibly from the same node) in rounds r and t respectively such that $r \leq t$.*
  - *(Consistent length) If $t \geq r+\Delta$,* $\left|\mathsf{chain}^t\right| \geq |\mathsf{chain}^r|$*.*
  - *(Chain growth bounds) If $g_0 \cdot (t-r) \geq T$, then* $\left|\mathsf{chain}^t\right| - |\mathsf{chain}^r| \geq \lfloor g_0 \cdot (t-r) \rfloor$*. If $g_1 \cdot (t-r) \geq T$, then* $\left|\mathsf{chain}^t\right| - |\mathsf{chain}^r| \leq \lceil g_1 \cdot (t-r) \rceil$*.*
- *((T,μ)-Chain-Quality) If* chain *is honest, then for any T consecutive blocks, at least μT blocks were mined by honest nodes.*

Security of Nakamoto's protocol [132]. We recall the main properties for Nakomoto's PoW protocol. Consider Nakamoto's protocol $\Pi_{\mathrm{nak}}(p)$ with hardness parameter $p(\cdot)$. Let α be the probability that some honest nodes mines a block in one round and let ν be the expected number of adversarial blocks mined in one round. Different quantities are considered for honest and adversarial nodes since any honest chain can grow by at most one block in any round while an adversary, who can query the H sequentially can append more than one block. Then, $\alpha = 1-(1-p(\kappa))^{(1-\beta)qn}$ and $\nu = \beta qnp(\kappa)$ [132][5]. Define $\alpha' = \frac{\alpha}{1+\alpha\Delta}$. Define a Nakamoto-compliance predicate, denoted by $\Gamma^{p}_{\mathrm{nak}}(n,\beta,\Delta) = 1$ if for all

[5] We consider q queries to H per round for each node whereas [132] considers a single query.

κ, there is some $\lambda > 1$ such that $\alpha(1-2(\Delta+1)\alpha) \geq \lambda\nu$. Then, in any environment that satisfies the Nakamoto-compliance predicate, or equivalently, for a sufficiently small hardness parameter, any $T_0 = \omega(\log\kappa)$, and any constant $\varepsilon > 0$, $\Pi_{\text{nak}}(p)$ satisfies T_0-consistency; (T_0, g_0, g_1)-chain-growth where $g_0 = (1-\varepsilon)\alpha'$, $g_1 = (1+\varepsilon)qnp$; and (T_0, μ) chain quality where $\mu = 1-(1+\varepsilon)\frac{\nu}{\alpha'}$. Note that all these quantities are functions of κ.

Proof simplifications. To make our proofs cleaner, we will make the following standard simplifications.

- (Simplification 1) (Mining is much slower than block propagation.)

  When the hardness parameter is sufficiently small, (e.g., when pq is small compared to $1/n$), we have $\alpha \approx q(1-\beta)np$. Therefore, by the union bound, the probability that no honest block is mined in Δ rounds is at most $(\Delta)\alpha$. If this is small, then with overwhelming probability, no other honest blocks are mined in the time it takes for an honestly mined block to be propagated to other nodes. This is reasonable since network propagation in Bitcoin is a few seconds, while a new block is mined roughly once every 10 minutes. For sufficiently small hardness parameter $p(\cdot)$, Pass et al. [132] showed that this happens with overwhelming probability in their proofs. We note that some prior works (eg., Prism [28]) also directly make a similar assumption.

- (Simplification 2) When the hardness parameter is sufficiently small, $\frac{\nu}{\alpha'}$ is well approximated by $\frac{\beta}{1-\beta}$, and therefore $\mu = 1-(1+\varepsilon)\frac{\nu}{\alpha'}$ is well approximated by $1-(1+\varepsilon)\frac{\beta}{1-\beta}$. Then, for any $\beta < \frac{1}{3}$, there will exist some ε_0 such that Π_{nak} satisfies chain quality with $\mu > \frac{1}{2}$. Specifically, for any $\epsilon > 0$ and $\beta = 1/3 - \epsilon$, let $0 < \varepsilon_0 < 1 - \frac{2+3\epsilon}{2-\epsilon}$. Then, Π_{nak} satisfies (T_0, μ_0)-chain quality where $T_0 = \omega(\log\kappa)$ and $\mu_0 > \frac{1}{2}$. In other words, in any T_0 consecutive blocks, the majority of blocks are mined by honest miners. In our proofs, we will directly use majority-chain-quality when $\beta < \frac{1}{3}$.

3.9 Connecting Aequitas Finalization to Voting Theory.

We start by describing a simpler static version of the finalization problem.

Problem 2. *(Strong (resp. Weak) Static Fair-Order Finalization) Consider an admissible $\Lambda = (\Upsilon, \mathcal{L})$ as input. Output a list* OutputList *containing all valid messages in $\mathcal{L}$ such that m is ordered before (resp. no later than) m' in* OutputList *if m is globally preferred over m' w.r.t. Λ, or equivalently, $m' \lhd^{\Lambda} m$.*

3.9.1 Connections to voting theory

It is easy to draw parallels between the Static Fair-Order Finalization problem (Problem 2) and majority voting systems, in which voters rank a set of candidates. In such a voting system, a candidate A is preferred over a candidate B if a majority of voters prefer A over B. The goal of a voting protocol now is to find the winning candidate(s), or more generally rank all candidates according to voter preferences. For our purpose, we are essentially replacing candidates with transactions and voters with consensus nodes. We note that Problem 2 also generalizes the condition when one transaction is preferred over another. Instead of a simple majority, we parameterize preference using a order-fairness parameter γ (see Definition 13). We also allow some of the "voting preferences" to be adversarial, which is not something usually considered in voting theory.

Condorcet cycles and Smith sets. One standard choice for selecting the winner(s) in a majority voting election is the Smith criterion [13]. The Smith set is the smallest possible set of candidates that are preferred over other candidates not in the set. When a Condorcet winner exists (i.e., there are no Condorcet cycles), it is always the sole member of the Smith set. By a Condorcet cycle, we mean a sequence of candidates (or messages), where the global preference relation is non-transitive. Such cycles can exist even when individual orderings are transitive due to the Condorcet paradox [8, 104].

Definition 17 ((Majority) Smith Set)**.** *The Smith set is the smallest set $\mathcal{S}$ that satisfies the condition that for all $x \in \mathcal{S}$ and $y \notin \mathcal{S}$, x is preferred over y in a majority of orderings.*

Static Finalization Algorithm **StaticFinalize**

Input: $\Lambda = (\Upsilon, \mathcal{L})$ where $\Upsilon = (n, f, \gamma)$ is admissible and $\mathcal{L} = [\mathsf{List}_1, ..., \mathsf{List}_n]$.

Notation: Let $\mathcal{V}$ be the set of valid messages in the input lists.

Algorithm Description.

1. Construct an empty graph $G = (V, E)$, where V denotes the set of vertices, and E denotes the set of edges. For all $m \in \mathcal{V}$, add a vertex labeled m to G.
2. For all $m, m' \in \mathcal{V}$, if $m' \lhd^{\Lambda} m$, then add the edge (m, m') to G.
3. Let G_{con} be the condensation graph of G.Output its topological sorting.

Figure 3-7: Static Finalization Algorithm

The Smith set can be computed using standard graph based algorithms. By representing global preferences as directed edges in a graph (e.g., $x \to y$ if x is globally preferred over y), computing the Smith set is now equivalent to finding the strongly connected component with no incoming edges from outside of the component. The Smith set is therefore, sometimes referred to as the "top cycle." It can be found by using variations of the Floyd-Warshall algorithm or Tarjan's algorithm (see [59] for common graph algorithms). Similarly, all candidates can be ranked by topologically sorting the strongly connected components.

We note that it is not particularly difficult to compute the fair ordering in the static finalization problem. Indeed, the same techniques for computing the Smith set can be used to find the fair ordering for transactions. We detail the algorithm in Figure 3-7. When no Condorcet cycles exist in $\mathcal{L}$, this protocol solves the strong static finalization problem; otherwise, it solves the weak variant.

3.9.2 Remarks on Aequitas

The main technical contribution of the Aequitas finalization algorithm is generalizing these techniques to solve a streaming version of the problem. KZGJ showed an impossibility result in the permissioned setting for the stronger receive-order-fairness property (where tx must be ordered before tx$'$) except in very specific settings. Informally, this arises due the Condorcet paradox [8] from voting theory literature. As a simple example, consider three nodes that receive transactions in the order $[x,y,z]$,$[y,z,x]$ and $[z,x,y]$ respectively. Note that each of "x before y", "y before z" and "z before x" is true for a majority of the nodes. This results in a cyclic or intransitive ordering dependency. Batch-order-fairness sidesteps this issue by allowing such paradoxical orderings to be output at the same time. Note that transactions inside a batch can still be totally ordered; their ordering just won't be taken into account for unfairness. This impossibility also carries over to the permissionless setting.

Aequitas finalization. We describe the Aequitas finalization technique from KZGJ . Given the n input orderings from the protocol nodes, the Aequitas finalization stage first builds a dependency graph of transactions where edges correspond to global preferences and computes its condensation graph by collapsing the strongly-connected-components. From the condensation graph, the transactions in a vertex v are delivered in one of two ways:

1. v is a source vertex and for all other vertices v', there is a path from v to v'
2. v is a source vertex and for any other vertex v' without a path from v, there is a common descendant of v and v' and either v has more descendants or v,v' have the same number of descendants but v is preferred in a previously agreed upon ordering relation (e.g., alphabetical ordering).

Intuitively, a transaction can be output if its strongly connected component can no longer change due to other transactions and it has no incoming edge. KZGJ showed that this finalization step is able to realize batch-order-fairness . We also show in Section 3.11.1 that it solves our streaming finalization formulation.

Remark 4 (Receive-Order-Fairness)**.** KZGJ *provided a synchronous permissioned protocol that achieves receive-order-fairness when the synchrony parameter in the external network (between clients and nodes) was* $\Delta_{\mathsf{ext}}=1$*. This is in part because an adversary can no longer claim to have received transactions too far apart from honest users (otherwise they would be easily detected), thereby resulting in no Condorcet cycles. For our purpose, since the adversary's orderings become intermingled with the honest ones through the PoW mining, attempting to realize receive-order-fairness would require rewinding PoW chains which might lead to other complications.*

Although the impossibility result for receive-order-fairness from KZGJ *when* $\Delta_{\mathsf{ext}} > 1$ *(and the adversary is allowed to corrupt at least one party) carries over to the permissionless setting, we leave it as an open problem to resolve whether receive-order-fairness is achievable in the permissionless setting when* $\Delta=1$*.*

Remark 5 (Minimally-relaxive batch-order-fairness)**.** *We can also show that although* $\mathsf{Aequitas}(\cdot)$ *solves the weak fairness of ordering property, the version it satisfies is in some sense a minimally relaxed form of the strong fairness of ordering property. Specifically, when* $m' \lhd^{\Lambda} m$*,* m *will be ordered before* m' *except when there is a Condorcet cycle that contains both* m *and* m'*. In other words,* m *and* m' *will be output at the same index only when a paradoxical ordering prevents them from being ordered any other way. To see why this holds, recall that the Aequitas finalization protocol from* KZGJ *outputs* m *and* m' *together if an only if they are part of the same strongly-connected-component in the dependency graph. Consequently, in such a case, there is a path from* m *to* m' *and from* m' *to* m *which creates a cyclic ordering dependency.*

Remark 6 (Remark on Liveness.)**.** *The Aequitas protocol from* KZGJ *achieves (conventional) liveness when no cycles can exist in the dependency graph or if the cycles cannot extend for across some bounded time interval. One such setting is when the external network (in the permissioned setting) has* $\Delta_{\mathsf{ext}}=1$*. When reducing the problem in the permissionless setting to running* $\mathsf{Aequitas}(\cdot)$ *on the* d *lists, we emphasize that this no longer corresponds to requiring* $\Delta=1$ *in our permissionless setting. Instead, this is the assumption that the sequence number of any given transaction in all* d *lists differs by at most 1. In our protocols, this turns out to be false due to potential adversarial influence on all* d *input lists. Fortunately, when* $\Delta=1$ *(in the permissionless environment), we can still show that any cycle that exists in the dependency graph of the* d *lists cannot extend for too long (Lemma 18). This allows our protocols to achieve conventional liveness.*

3.10 Additional Protocol Descriptions

Figures 3-8 and 3-10 contain a pictorial representation of our single-chain protocol $\Pi^{\kappa,T}_{\text{mod}}(p)$ and our multi-chain protocol $\Pi^{\kappa,g}_{\text{multi}}(p)$ respectively.

3.10.1 Fruitchains version of Π_{mod}

We briefly sketch a version of Π_{mod} that uses the Fruitchains protocol from [133] as the underlying protocol instead of Π_{nak}. The proofs for security will almost exactly mirror the proofs for Π_{mod} and therefore, we only provide an outline here.

Background on Fruitchains. In the Fruitchains protocol, nodes create two types of objects: fruits and blocks. Fruits are the ones that actually contain transactions while blocks are used to order fruits (and thereby order transactions). Fruits are required to "hang" from recent blocks, where recency is defined using a recency parameter Y. To mine both fruits and blocks, nodes use the same PoW mining process (using the 2-for-1 mining technique [91]) which using a random oracle with output size 2κ. A block is said to be mined if the first κ bits of the hash are less than the mining hardness threshold D_p, while a fruit is said to be mined if the last κ bits of the hash are less than some other threshold D_f. A block will be propagated to the rest of the network as part of the chain it gets mined in, whereas a fruit can be simply broadcast. The recency parameter mandates only recent fruits—those that hang from one of the last $Y\kappa$ blocks—are mined into a block.

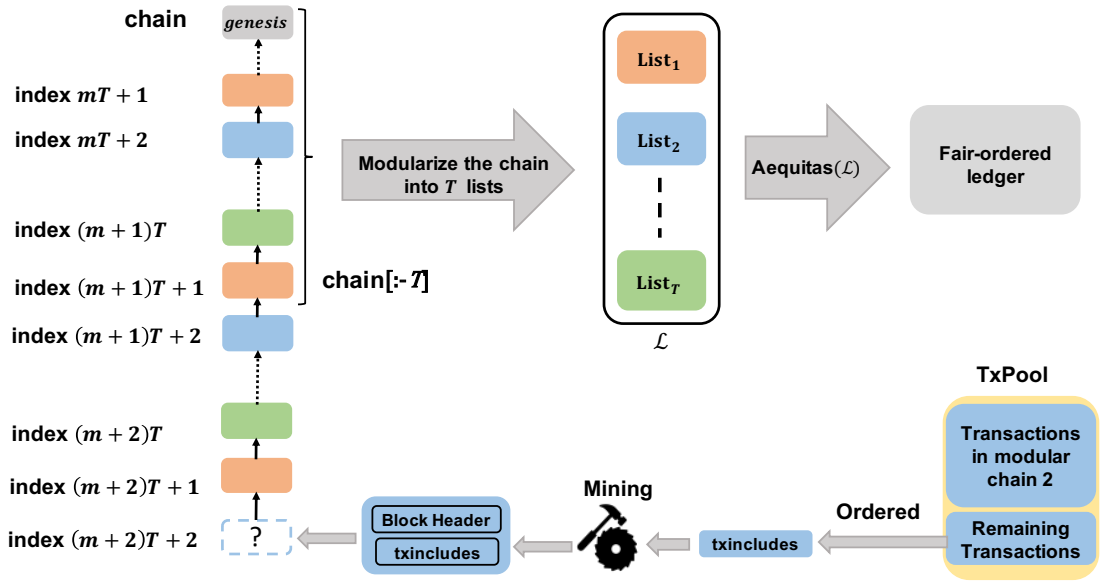


Figure 3-8: Pictorial representation of our single-chain protocol Π_{mod}. Π_{mod} mines a single Nakamoto PoW chain which is modularized into T semantic chains represented by different colors. The underlying Nakamoto PoW chain satisfies T-prefix-consistency and $(T,\mu>\frac{1}{2})$-chain-quality. Aequitas$(\cdot)$ is run on the confirmed parts (in the PoW chain) of these chains $\mathcal{L}=[\text{List}_1,\ldots,\text{List}_T]$ to extract the fair transaction ordering.

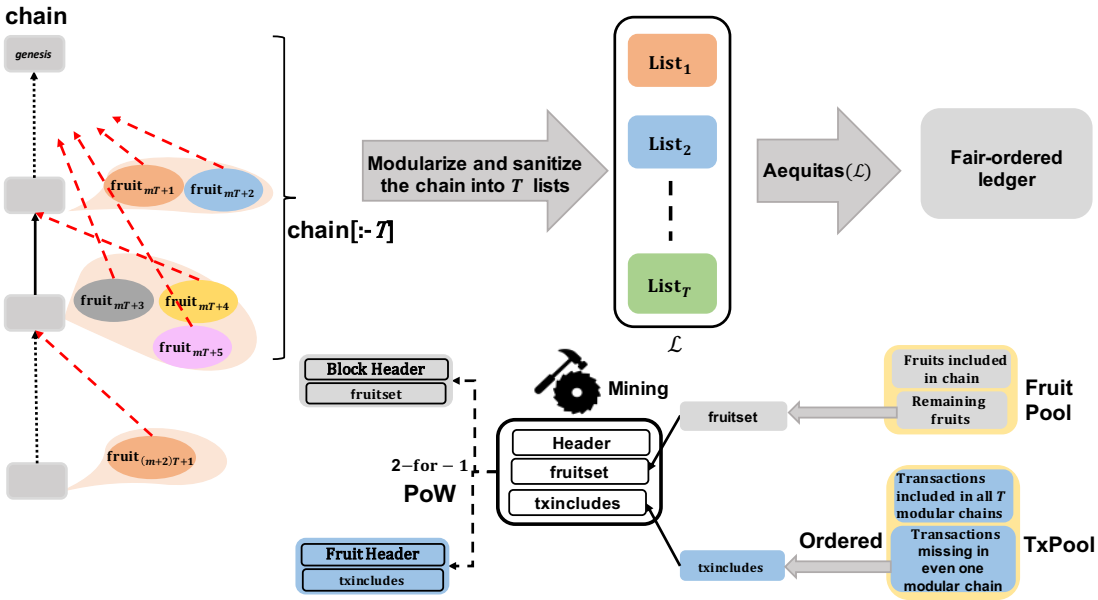


Figure 3-9: Pictorial representation of the fruitchain version of single-chain protocol Π_{fruit}. Π_{fruit} mines either fruit or a block in a single Nakamoto PoW chain using the 2-for-1 mining technique. The fruits are modularized into T semantic chains represented by different colors. The underlying Nakamoto PoW chain satisfies T-prefix-consistency and $(T,\mu>\frac{1}{2})$-chain-quality. Aequitas$(\cdot)$ is run on the confirmed parts (in the PoW chain) of these chains $\mathcal{L}=[\text{List}_1,\ldots,\text{List}_T]$ to extract the fair transaction ordering.

Fair ordering protocol description. Consider the protocol $\Pi_{\text{fruit}}^{\kappa,T,Y}$ that mines a single fruitchain with recency parameter Y. T is the modular parameter for the protocol where $T=\Theta(\kappa)$ will be based

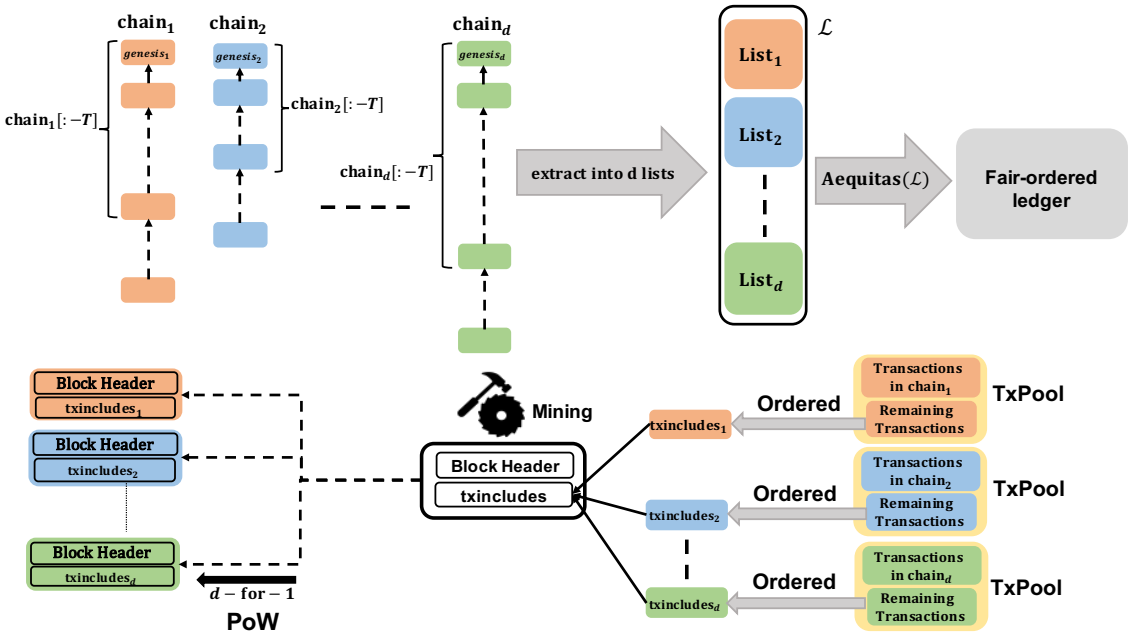


Figure 3-10: Pictorial representation of our multi-chain protocol Π_{multi}. Π_{multi} mines $d = \Theta(\kappa)$ Nakamoto PoW chains simultaneously using the d-for-1 mining technique. The underlying chains all satisfy T-prefix-consistency and $(T, \mu > \frac{1}{2})$-chain-quality. $\mathsf{Aequitas}(\cdot)$ is run on the confirmed parts of each these chains, i.e., on $\mathcal{L} = [\mathsf{List}_1, ..., \mathsf{List}_d]$, to extract the fair transaction ordering.

on the consistency parameter for the underlying fruitchain. For a given block $\mathsf{chain}[b]$, the miner of the block indexes the fruits starting from one more than the index of the last fruit in the parent block ($\mathsf{chain}[b-1]$). Fruits in the same index modulo T will be considered to be semantically or logically connected. For example, fruits at indices 1, $T+1$, $2T+1$, and so on will be part of the same "modular chain." This divides the single PoW chain into T modular chains. We modularize the chains using fruits instead of blocks since the chain-quality property for Fruitchains is given in terms of fruits.

Now, in the same structure as Π_{mod}, from the T modular chains, T separate transaction orderings will be deduced which will be used as input for the $\mathsf{Aequitas}(\cdot)$ algorithm to retrieve the final transaction ordering. $\mathsf{Aequitas}(\cdot)$ will be run on the confirmed parts of the fruitchain i.e., with the last T blocks cut off, where $T = \Theta(\kappa)$. This will extract the fair ordering from the fruitchain. Note that the consistency parameter here is larger than the $\omega(\log\kappa)$ requirement for the standard Π_{mod}. Figure 3-9 contains the pictorial representation of Π_{fruit}.

Security argument sketch. Pass and Shi [133] showed that there is some recency parameter $Y = \Theta(1)$ and some $\kappa_f = \Theta(\kappa)$ such that the Fruitchains protocol satisfies κ_f-prefix-consistency, and $(\frac{5\kappa_f}{\delta}, \mu = (1-\delta)(1-\beta))$-chain-quality for any $0 < \delta < 1$. For $\beta < \frac{1}{2}$, there exists a $\delta = \delta_0 > 0$ such that $\mu > \frac{1}{2}$. This means that for $T = \frac{5\kappa_f}{\delta_0}$, Fruitchains satisfies both T-consistency and $(T, \mu > \frac{1}{2})$-chain-quality. Note that this T is $\Theta(\kappa)$.

Now, exactly the same proofs for Π_{mod} can be used to show that $\Pi_{\text{fruit}}^{\kappa,T,Y}$ is both dependency-complete and dependency-faithful when $\gamma - (\beta + \tau) > \frac{1}{2}$ which will directly imply order-fairness and liveness. Note that consistency once again comes for free from the consistency of the underlying fruitchain.

3.11 Deferred Proofs

In this section, we provide the proofs deferred from the main body of the paper. For convenience, whenever appropriate, we write the lemma statements again.

3.11.1 Proofs from Section 3.4

Lemma 19. *For an admissible* Υ, $\mathsf{Aequitas}_\Upsilon(\cdot)$ *solves the weak streaming fair-order finalization problem.*

Proof. First, from the self-consistency of the Aequitas protocol, we can infer that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $\mathsf{Aequitas}_\Upsilon(\mathcal{L}) \preccurlyeq \mathsf{Aequitas}_\Upsilon(\mathcal{L}')$. Furthermore, $\mathsf{Aequitas}_\Upsilon(\cdot)$ satisfies the weak fairness of ordering property since the Aequitas protocol from KZGJ satisfies batch-order-fairness in the permissioned setting. □

We prove a more general version of the player replaceability lemma next.

Lemma 20 (Player Replaceability)**.** *Consider an* $(\mathcal{A},\mathcal{Z})$ *that satisfies* $(n,\beta,\Delta,\cdot,\cdot)$*-permissionless execution, an order-fairness parameter* γ*, an admissible* $\Upsilon = (d,d\beta',\gamma)$ *for some* $0 \leq \beta' \leq 1$*. Suppose that* Π *is a protocol as follows: For any node* N *that is honest in round* r*,* N *outputs* $\mathcal{L}_N^r = [\mathsf{List}_1^r,\ldots,\mathsf{List}_d^r]$ *to* $\mathcal{Z}$ *that satisfies the following:*

1. *If* N *is also honest in round* $r-1$*, then* $\mathcal{L}_N^{r-1} \preccurlyeq \mathcal{L}_N^r$*.*
2. *For any pair of transactions* $(\mathrm{tx},\mathrm{tx}')$ *that satisfies the antecedent of the (permissionless)* γ*-order-fairness definition (Definition 12), if either* tx *or* tx' *is present in all* d *lists in* $\mathcal{L}_N^r$*, then* tx *will be* γ*-globally preferred to* tx' *w.r.t.* $\Lambda = (\Upsilon,\mathcal{L}_N^r)$*. In other words, the edge* $(\mathrm{tx},\mathrm{tx}')$ *will be in the dependency graph* $\mathsf{Graph}(\mathcal{L})$*.*

Suppose further that an algorithm $F(\cdot)$ *satisfies the streaming strong (resp. weak) fair-order finalization problem for* Υ *such that the* linearize *function of* Π *is defined to be* $F(\mathcal{L}_N^r)$*. Then,* Π *satisfies* γ*-receive-order-fairness (resp.* γ-batch-order-fairness *) w.r.t.* $(\mathcal{A},\mathcal{Z})$*.*

Proof. For any pair $(\mathrm{tx},\mathrm{tx}')$ that satisfies the antecedent of the (permissionless) γ-order-fairness definition, we are given that Π is such that tx will be γ-globally preferred to tx' in the d lists that represent the virtual parties. Now, since $F(\cdot)$ satisfies the strong (resp. weak) fairness of ordering property, the final output ledger of an honest node in the given round, will contain tx before (resp. no later than) tx'. □

3.11.2 Proofs from Section 3.5.1

Lemma 21. *Consider* $(n,\beta,\Delta = 1,\tau,R)$ *and suppose that a protocol* Π *is dependency-faithful (Property 2). Suppose that some honest node in round* r *outputs* $\mathcal{L}$ *where* tx *and* tx' *are in the same Condorcet cycle. Let* r,r' *be the first round that some honest node has received* tx *and* tx' *respectively. Then* $r = r'$*.*

Proof. Let $(\mathrm{tx}_1,\mathrm{tx}_2,\ldots,\mathrm{tx}_l)$ be a Condorcet cycle in $\mathcal{L}$ and suppose that tx and tx' are in this cycle. Let round r_{first} be the first round that any one of $\{\mathrm{tx}_1,\ldots,\mathrm{tx}_l\}$ was received by an honest node. We can assume tx_l was first received by an honest node at round r_{first} without loss of generality by rotating the tuple cyclically. This means that any honest node that spawned no later than $r_{\text{first}}+1$, received tx_l either in round r_{first} or $r_{\text{first}}+1$, and any honest node that spawned in round $r' > r_{\text{first}}+1$ received tx_l in round r'.

Now, suppose that tx_{l-1} was first received by an honest node at round r_{l-1} and suppose for the sake of contradiction that $r_{l-1} \geq r_{\text{first}}+1$. This means that any honest node that spawned no later than r_{l-1} had already received tx_l (i.e., either tx_l before tx_{l-1} or tx_l and tx_{l-1} in the same round) and any node that spawned after received both tx and tx' at the same time. However, since we have $\mathrm{tx}_l \lhd \mathrm{tx}_{l-1}$, this contradicts the dependency-faithfulness of Π since there needs to be some honest node that received tx before tx'. Furthermore, since no transaction in $\{\mathrm{tx}_1,\ldots,\mathrm{tx}_l\}$ was received before r_{first}, this means that tx_{l-1} was first received by an honest node in round r_{first}.

Continuing in the same fashion, the same argument shows that any transaction in $\{\mathrm{tx}_1,\ldots,\mathrm{tx}_l\}$, including tx and tx', was first received by an honest node in round r_{first}.

Note that also applies to the case where some of the transactions are adversarial (i.e. first seen by the network as part of a block rather than transaction gossip). Recall that the ledger synchrony assumption of our network ensures that transactions are provided as input to all honest nodes if some honest node outputs them to the environment. □

3.11.3 Proofs from Section 3.5.2

We start by showing that Π_{mod} is dependency-complete which will allow us to conclude order-fairness.

Lemma 22. *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta,\gamma,T,\tau,R)=1$. *Then,* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *is* γ*-dependency-complete for* (n,β,Δ,τ,R) *environments.*

Proof. Consider any $(\mathcal{A},\mathcal{Z})$ that satisfies (n,β,Δ,τ,R)-permissionless execution. Consider a pair of transactions (tx,tx') that satisfies the antecedent of the order-fairness definition. For a given chain, suppose that $\mathsf{chain}[b]$ is the first block that contains either tx or tx′ and that $|\mathsf{chain}|\geq b+2T-1$. Note that all blocks in $\mathsf{chain}[b,b+T-1]$, and consequently all modular chains, will include either tx or tx′. Now, since $\gamma-(\beta+\tau)>\frac{2}{3}$ and $\mathcal{A}$ is (τ,R)-respawning, there are more than $\frac{2n}{3}$ nodes that received tx before tx′ and are honest until all modular chains contain either tx or tx′ (i.e., until $\mathsf{chain}[b+T-1]$ has been confirmed). Let H be the set of these nodes. Now, from the chain-quality of $\Pi_{\text{nak}}(p)$, in the range $\mathsf{chain}[b,b+T-1]$, more than $\frac{T}{2}$ of these blocks are mined by the nodes in H. In other words, more than half of the blocks in $\mathsf{chain}[b,b+T-1]$ will contain tx before tx′. Since these blocks have been confirmed in chain, more than half of the lists in $\mathcal{L}=\mathsf{modularize}(\mathsf{chain}[:-T],T)$ will contain tx before tx′. Since $\Upsilon=(T,\lceil\frac{T}{2}\rceil-1,1)$ is admissible, tx is globally preferred to tx′ w.r.t. $(\Upsilon,\mathcal{L})$, i.e. $\text{tx}'\lhd^{(\Upsilon,\mathcal{L})}\text{tx}$.We conclude that $\Pi^{\kappa,T}_{\text{mod}}$ is dependency-complete. □

Theorem 8 (Order-Fairness of Π_{mod})**.** *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta,\gamma,T,\tau,R)=1$. *Then,* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *satisfies* γ-batch-order-fairness .

Proof. This is a straightforward combination of Fact 19, Lemma 20, and Lemma 22. Note that for the simplest case of $\gamma=1$, we require $\beta+\tau<1/3$. □

Next, we show that Π_{mod} is dependency-faithful in the following lemma.

Lemma 23. *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta,\gamma,T,\tau,R)=1$. *Then,* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *is dependency-faithful for* (n,β,Δ,τ,R) *environments.*

Proof. Consider any $(\mathcal{A},\mathcal{Z})$ that satisfies (n,β,Δ,τ,R)-permissionless execution. Suppose that a node N is honest in round r, holds chain, and outputs $\mathcal{L}=\mathsf{modularize}(\mathsf{chain}[:-T],T)$ to $\mathcal{Z}$. Note that $\mathcal{L}$ only contains the confirmed part of each modular chain. Let $\Upsilon=(T,\lceil\frac{T}{2}\rceil-1,1)$ and $\Lambda=(\Upsilon,\mathcal{L})$. Suppose that $\text{tx}'\lhd^{\Lambda}\text{tx}$, and let $\mathsf{chain}[b]$ be the first block that contains either tx or tx′. Then, the range $\mathsf{chain}[b,b+T-1]$ will determine the ordering between tx and tx′ in each of the T modular chains. Since $\text{tx}'\lhd^{\Lambda}\text{tx}$, more than $\frac{T}{2}$ of these blocks order tx before tx′. But, since $\beta<\frac{1}{3}$ (from $\gamma-(\beta+\tau)>\frac{2}{3}$), the chain-quality property of $\Pi_{\text{nak}}(p)$ implies that less than half of the blocks in the range $\mathsf{chain}[b,b+T-1]$ are mined by an adversary. This means that there is at least 1 block that contained tx before tx′ which was mined by an honest node. Therefore, some honest node received tx before tx′ from $\mathcal{Z}$. We conclude that $\Pi^{\kappa,T}_{\text{mod}}$ is dependency-faithful. □

We can now use the chain-growth of Π_{nak} to bound the liveness for Π_{mod} in the following theorem.

Theorem 9 (Liveness of Π_{mod})**.** *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta=1,\gamma,T,\tau,R)=1$, *and suppose that* $\Pi_{\text{nak}}(p)$ *satisfies* (T,g_0,g_1)*-chain-growth. Then* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *satisfies* $(5T+2)/(2g_0)$*-liveness.*

Proof. Consider any transaction tx that is first input in round r. By the time more than $\frac{5T}{2}$ blocks are mined, all semantic chains will contain tx in the confirmed part of the chain as well as any transactions that are in the same cycle as tx. Furthermore, any transaction that is first received by an honest node at $r+2$ will be a descendant of tx (as well as any other tranasction without an edge to/from tx) in the dependency graph of an honest node. Note that such a transaction will also be in the confirmed part of the chain. Consequently, by this time $\mathsf{Aequitas}(\cdot)$ will have delivered tx.

Now, we note that for a round r' such that $r'-r\geq\frac{5T+2}{2g_0}$, we have $g_0(r'-r)\geq\frac{5T}{2}+1$, which implies that tx will be output by round r' by all honest nodes. Consequently, Π_{mod} satisfies $\frac{5T+2}{2g_0}$-liveness. □

This allows us to conclude the main security result for Π_{mod} which we restate here.

Theorem 10. *Consider parameters such that* $\Gamma^{p}_{\text{mod}}(n,\beta,\Delta=1,\gamma,T,\tau,R)=1$. *Then,* $\Pi^{\kappa,T}_{\text{mod}}(p)$ *satisfies consistency, R-liveness, and* γ-batch-order-fairness *for* (n,β,Δ,τ,R) *environments.*

Remark 7 ($\beta<\frac{1}{2}$ using Fruitchains)**.** *We can handle minority corruption by using Fruitchains [133] as the underlying consensus protocol. Fruitchains achieves optimal mining-fairness, which means that for any* $\beta<\frac{1}{2}$*, and a sufficiently long period* T*, it has* $(T,\mu_0>\frac{1}{2})$*-chain-quality. Consequently, by using Fruitchains, our construction can tolerate* $\beta+\tau<\frac{1}{2}$ *corruption and still achieve order-fairness (for* $\gamma=1$*). We provide a brief sketch of the protocol in Section 3.10.1. Since we require only the appropriate cross-chain-quality parameters, the security proofs for such as protocol will almost exactly mirror the ones for* Π_{mod}*.*

Remark 8 (Other longest-chain protocols)**.** *Note that since our proofs only utilize the abstract blockchain properties of the underlying PoW chain, our single-chain construction can also be layered on top of other longest-chain protocols that satisfy the chain-quality property (e.g., Proof-of-Stake protocols like [107]).*

3.11.4 Proofs from Section 3.5.3

We start by showing that cross-chain-quality implies both dependency-completeness and dependency-faithfulness for Π_{multi}.

Lemma 24. *Consider* $(n,\beta,\Delta=1,\tau,R)$*, and d, and define* $\Pi^{\kappa,g}_{\text{multi}}(p)$ *as before. Suppose that for all* $(\mathcal{A},\mathcal{Z})$ *that support* $(n,\beta,\Delta=1,\tau,R)$*-permissionless-execution,* $\Pi^{\kappa,d}_{\text{multi}}(p)$ *satisfies cross-chain-quality (except for a negligible number of views), then it is also 1-dependency-complete and dependency-faithful.*

Proof.

1. (Dependency-Completeness) Suppose that $\Pi^{\kappa,g}_{\text{multi}}(p)$ satisfies cross-chain quality, and suppose for contradiction, that it is not dependency complete. Therefore, there is a pair of transactions $(\text{tx}_1,\text{tx}_2)$ that satisfies the antecedent of γ-order-fairness but an honest node outputs $\mathcal{L}$ with $\text{tx}_2 \lhd^{\Lambda} \text{tx}_1$ where $\Lambda=(\Upsilon,\mathcal{L})$ and $\Upsilon=(g,\lceil\frac{g}{2}\rceil-1,1)$. Now, for each chain, we consider 2 possibilities for the placement of the two transactions: either tx_1 is included before tx_2 or tx_2 is included before tx_1 (The case of both included at the same place is not relevant for our purpose). Furthermore, there are 4 possibilities for mining of the blocks containing tx_1,tx_2: both honest, both adversarial, only tx_1 honest, only tx_2 honest. We use the following notation to denote these 8 combinations: (H_1,A_2) represents that tx_1 was included before tx_2 and the block containing tx_1 was mined by an honest node while the block containing tx_2 was mined by an adversarial node. Let $\mathsf{num}(H_1,A_2)$ be the number of chains that have this combination. Since, $\text{tx}_2 \lhd^{\Lambda} \text{tx}_1$, more than $\frac{g}{2}$ should have included tx_2 before tx_1. Therefore,

$$\mathsf{num}(H_2,H_1)+\mathsf{num}(H_2,A_1)+\mathsf{num}(A_2,H_1)+\mathsf{num}(A_2,A_1)>\frac{g}{2}$$

   But since all honest nodes received tx_1 before tx_2 during the initial propagation, and $\Delta=1$, any honest node that spawns later will receive tx_1 as soon as it spawns and therefore cannot have received tx_2 before tx_1. This means that, $\mathsf{num}(H_2,H_1)$ and $\mathsf{num}(H_2,A_1)$ will both be zero since no honest node will include tx_2 before tx_1. Therefore,

$$\mathsf{num}(A_2,H_1)+\mathsf{num}(A_2,A_1)>\frac{g}{2}$$

   But this contradicts cross-chain-quality since the adversary was able to mined tx_2 in more than half of the chains. This completes the proof.

2. (Dependency-Faithfulness) Suppose that $\Pi^{\kappa,g}_{\text{multi}}(p)$ satisfies cross-chain quality, and suppose for contradiction, that it is not dependency-faithful. Therefore, there is an honest node that outputs

$\mathcal{L}$ with $\mathrm{tx}_2 \lhd^{\Lambda} \mathrm{tx}_1$ where $\Lambda = ((g, \lceil \frac{g}{2} \rceil - 1, 1), \mathcal{L})$ but there is no honest node that received tx_1 before tx_2. Now, using the same notation from the dependency-completeness argument, we have

$$\mathsf{num}(H_1,H_2)+\mathsf{num}(H_1,A_2)+\mathsf{num}(A_1,H_2)+\mathsf{num}(A_1,A_2) > \frac{g}{2}$$

Since there is no honest node that received tx_1 before tx_2 (including nodes that spawn later since $\Delta = 1$), both $\mathsf{num}(H_1,H_2)$ + $\mathsf{num}(H_1,A_2)$. Therefore, $\mathsf{num}(A_1,H_2) + \mathsf{num}(A_1,A_2) > \frac{g}{2}$, which once again will contradict cross-chain-quality for tx_1.

□

We reserve Section 3.12 for the proof that Π_{multi} satisfies the cross-chain-quality property. Equipped with this, we can conclude the main security result for Π_{multi} which we restate here.

Theorem 11. *Consider parameters such that* $\Gamma^{p}_{\text{multi}}(n,\beta,\Delta=1,\gamma=1,\tau,R,d,W)=1$. *Then,* $\Pi^{\kappa,d}_{\text{multi}}$ *satisfies consistency,* (W,R)*-liveness, and* 1-batch-order-fairness *for* (n,β,Δ,τ,R) *environments.*

3.11.5 Proofs from Section 3.6

Theorem 12. *If* Π *satisfies* γ*-receive-order-fairness w.r.t.* $(\mathcal{A},\mathcal{Z})$ *with network parameter* Δ*, then it also satisfies* (2Δ)*-soft-ordering.*

Proof. The proof is straightforward. Suppose that a node N receives tx in round r. Then all honest nodes have received tx latest by round $r+\Delta$. Since node N received tx' at round later than $r+2\Delta$, that implies that tx' was received at all other honest nodes earliest at round $r+\Delta+1$. Therefore, $(\mathrm{tx},\mathrm{tx}')$ is such that all honest nodes received tx before they received tx', Since Π satisfies receive-order-fairness, tx will be ordered before tx' in the final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-ordering. Note that the same proof also works for fair-linearizability (from Section 3.2.3).

Now, if N sees no conflicting transaction by round $r+2\Delta$, then any conflicting transaction was received by an honest node at the earliest by round $r+\Delta+1$. Therefore, $(\mathrm{tx},\mathrm{tx}')$ is such that all honest nodes received tx before they received tx', Since Π satisfies receive-order-fairness, tx will be ordered before tx' in the final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-confirmation. □

3.11.6 Soft-Ordering as Hamiltonian paths

Earlier, we discussed how the soft-ordering property allows honest transactions to be quickly locally confirmed when there are no conflicting or double-spend transactions. This requires a notion of "conflicting transaction" which may not always be clear. Here, we show another angle for looking at soft-confirmation in terms of the possible orderings the final log could contain. In other words, we show how soft-confirmation allows nodes to narrow down possibilities for how the final ordered log might look, just based on their input orderings and long before the transactions are officially confirmed.

For this, suppose that a protocol Π satisfies δ-soft-ordering. Now, a node N constructs a graph $\mathcal{S}_N$ locally as follows, based only on the ordering in which it received transactions.

- For all transactions tx, a vertex labeled tx is added to the graph.
- For vertices tx and tx' in $\mathcal{S}_N$ such that N received tx earlier, an edge $(\mathrm{tx},\mathrm{tx}')$ is added if N received tx more than δ rounds before tx'. On the other hand, both the edges $(\mathrm{tx},\mathrm{tx}')$ and $(\mathrm{tx}',\mathrm{tx})$ are added if tx was received less than δ rounds before tx'.

In Figure 3-11, we show an example of how to construct this local graph.

Now, consider the final totally ordered ledger $\mathbb{L}$ for the protocol Π. In case of transactions output at the same index, (as in batch-order-fairness), a suitable transformation is used to fully order these

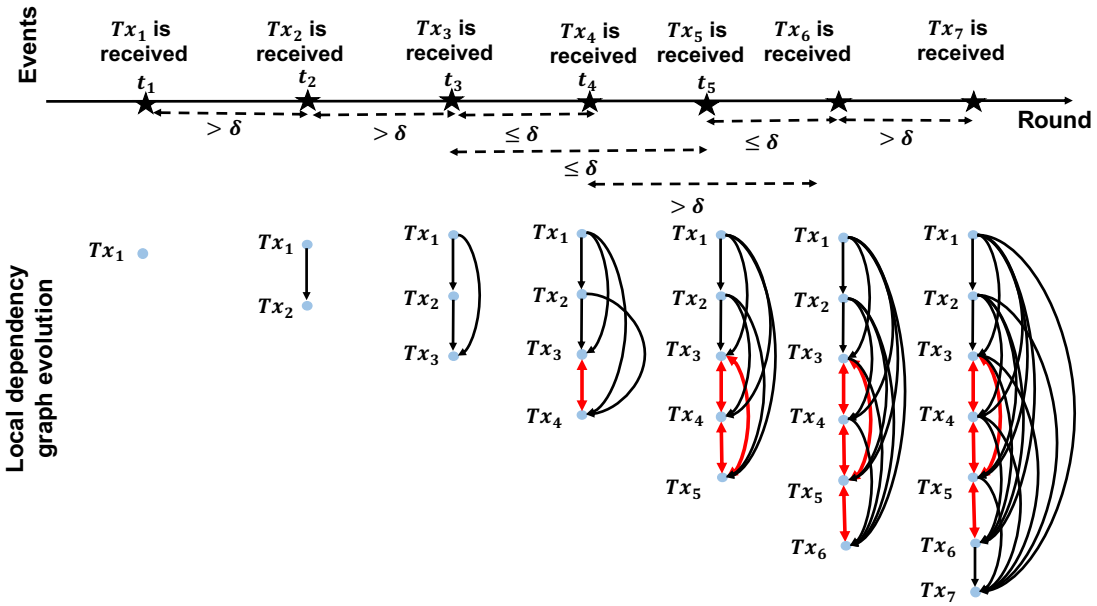


Figure 3-11: Pictorial representation of the construction of the local dependency graph for a protocol that satisfies δ-soft-ordering. The black-colored edges in the graph represent directed edges and the red-colored edges represent bidirectional edges.

transactions. Recall that for batch-order-fairness , transactions are still totally ordered even inside a batch, but orderings within a batch do not account for unfairness.

We now show that at any point, there is some Hamiltonian path of $\mathcal{S}_N$ that is a prefix of the final ledger. Recall that a Hamiltonian path of a connected graph is a path that visits all vertices exactly once.

Lemma 25. *Consider a protocol Π that satisfies δ-soft-ordering. Let $\mathbb{L}_{\mathsf{view}}$ be the final totally ordered ledger for a view* view*. Let $\mathcal{S}_N^r$ denote the previously defined graph for node N at round r. Then for any round r, there is a Hamiltonian path P of $\mathcal{S}_N^r$ that satisfies the following property $\mathcal{Q}$:*

- *Let V be the last vertex of P. Construct path P^* by removing from P all vertices that are reachable from V in $\mathcal{S}_N^r$. Informally, the removed transactions are the ones whose ordering is not yet fixed as their ordering could depend of transactions N receives shortly after.*
- *The transaction ordering given by P^* is a prefix of $\mathbb{L}_{\mathsf{view}}$.*

Proof. We ignore the negligible number of bad views (for example, where soft-ordering is not satisfied). Suppose for contradiction that no Hamiltonian path exists that satisfies the property $\mathcal{Q}$. Now consider some Hamiltonian path P and its corresponding P^* (we know this exists since $\mathcal{S}_N^r$ is complete). If P^* is empty, then it is trivially a prefix of $\mathbb{L}_{\mathsf{view}}$. So consider a non-empty P^*.

Now, since P^* is not a prefix of $\mathbb{L}_{\mathsf{view}}$, then the following holds: There is a transaction tx ordered before tx$'$ in $\mathbb{L}_{\mathsf{view}}$ such that either (1) tx is ordered after tx$'$ in P^*; or (2) P^* contains tx$'$ but not tx. We will proceed by a simple recursive argument on the number of such transaction pairs.

Suppose that there is only one pair (tx,tx$'$) that breaks the prefix property. If (1) is true, then either $\mathcal{S}_N^r$ contains a unidirectional edge (tx$'$,tx) or it contains edges in both directions. However, since Π satisfies δ-soft-ordering, the unidirectional edge would imply the ledger always contains tx$'$ earlier. Consequently, $\mathcal{S}_N^r$ needs to contain a bidirectional edge between the two transactions. This means that there is another path that simply flips the position of tx and tx$'$, and this would now be a prefix of $\mathbb{L}_{\mathsf{view}}$.

If (2) is true, then this means tx$'$ was received more than δ rounds before tx (since tx$'$ was present in P^* but tx wasn't). But this implies that $\mathbb{L}_{\mathsf{view}}$ must contain tx$'$ first which is a contradiction.

Now, given any path P_0, and the corresponding P_0^* that conflicts with $\mathbb{L}_{\mathsf{view}}$ in k transaction pairs, we can use the above technique to arrive at another Hamiltonian path P that satisfies the property $\mathcal{Q}$. At every step of the recursion, the number of conflicting transaction pairs will be reduced by at least 1, i.e., the recursion will take at most k steps

At the end of the recursion, we get a path P which satisfies $\mathcal{Q}$, and therefore contradicts our assumption. This completes this proof. □

3.12 Cross-chain-quality proof for Π_{multi}

We restate the cross-chain-quality property below. We will show that it holds, except for a negligible number of views.

Property 4 (Cross-Chain-Quality)**.** *For a given* view *and* tx*, let* $\mathcal{B}_{\text{tx}}$ *be the set of blocks that contain* tx *for the first time in the confirmed part (i.e.* $\mathsf{chain}_j[:-T]$*) of each of the* d *chains. Then, more than half of the blocks in* $\mathcal{B}_{\text{tx}}$ *were mined by honest nodes.*

Notation. Before proceeding with the proof, we start by introducing some useful notation from [28]. Let $f = npq$ denote the mining rate (as a function of κ) per round for one chain. Typically, in practice, the hardness parameter p is set as $p = \Theta(\frac{1}{\Delta qn})$ which would make $f = \Theta(1)$ in κ since we are using $\Delta = 1$. This will make our analysis cleaner. Recall that d is the number of parallel chains and β is the adversarial corruption threshold. We use the following random variables in our proofs. Note that the probabilities are all taken over $\mathsf{view} \leftarrow\!\!\$\, \mathsf{EXEC}^{\Pi_{\text{multi}}}(\mathcal{A}, \mathcal{Z}, \kappa)$ although for succinctness, we do not explicitly write this. For a chain i, and round r, we define:

- $X_i[r] = \begin{cases} 0, \text{ if no honest node mines a block in round } r \\ 1, \text{ if at least one honest node mines a block in round } r \end{cases}$
- $X_i[r_1, r_2] = \sum_{r=r_1}^{r_2} X_i[r]$.
- $Y_i[r] = \begin{cases} 1, \text{ if exactly one honest node mines a block in round } r \\ 0, \text{ otherwise} \end{cases}$
- $Y_i[r_1, r_2] = \sum_{r=r_1}^{r_2} Y_i[r]$.
- $Z_i[r]$ is the number of blocks mined by an adversarial node in round r.
- $Z_i[r_1, r_2] = \sum_{r=r_1}^{r_2} Z_i[r]$.
- $C[:\ell]$ is the prefix of the longest chain C until the level ℓ.
- $C[\ell]$ is the block at level ℓ of the chain C.

For an event Q, use $\mathbf{1}_Q$ to denote the indicator variable for when Q holds. That is, $\mathbf{1}_Q = 1$ when Q is true and 0 otherwise. We use the notation $a_+ = \max\{0, a\}$.

Following [28], $Z_i[r]$ follows a Poisson distribution, while $X_i[r]$ follows a Bernoulli distribution. Specifically, we have,

- $Z_i[r] \sim \text{Poisson}(\beta f)$
- $X_i[r] \sim \text{Bernoulli}(1 - e^{-(1-\beta)f})$

3.12.1 Proof structure.

In order to prove the cross-chain-quality property for Π_{multi}, abstractly we will construct an event $V_i[r]$ for each chain i at round r, which will guarantee that any transaction first seen in round r will be contained in a block mined by an honest node. Now, the probability in an arbitrary chain i at any round r, that $V_i[r]$ happens (as a function of β and f) is larger than 0.5, then we can set the number of chains $d = \Theta(\kappa)$, and use the Chernoff bound to argue that the probability that any transaction is mined in more than half the parallel chains by the adversary is negligible in κ. We elaborate on the main proof parts below.

Constructing $V_i[r]$. We start by defining the following random processes for all chains i and rounds $r > 0$.

$$S_i[r] = (S_i[r-1] + Z_i[r] - X_i[r])_+$$
$$L_i[r,t] = X_i[r,t] - Z_i[r,t].$$

We also set $S_i[0] = 0$. Now, if the adversary never reveals the blocks it mines, $S_i[r]$ captures the difference between the length of longest private adversarial chain i at round r, and the longest public chain held by honest nodes. The random process $L_i[r,t]$ represents the difference between the number of honest blocks and adversarial blocks mined in chain i from round r to round t.

Now, we define $V_i[r]$ for chain i at round $r > 0$ as the event that all of the following holds:

1. $S_i[r-1] = 0$
2. There exists a round $t' > r$ such that
   - $Z_i[r,t'] = 0$. In other words, the adversary was not able to mine a block from round r to t'.
   - $X_i[r,t'-1] = 0$. In other words, honest nodes were not able to mine any block from r until $t'-1$.
   - $Y_i[t'] = 1$. In other words, exactly one honest node mined a block in round t'.
   - $L_i[r+1,t] \geq 1 \quad \forall t > t'$. In other words, the number of honest blocks mined in chain i from round $r+1$ until round t is greater than the corresponding number of adversarial blocks in the same interval, for all $t > t'$.

We will call the first event $V_i^1[r]$ and the second event $V_i^2[r]$. We evaluate their probabilities separately since they are independent.

Honest block when $V_i[r]$ holds. In Section 3.12.2, we show that whenever $V_i[r]$ holds, any transaction tx that is first revealed (to honest nodes) at round r, in chain i, would be contained in a block mined by honest miner. Intuitively, there are two ways a transaction can be propagated:

- *Honestly propagated transaction*: A transaction that is input to all nodes (honest and adversarial) within $\Delta = 1$ rounds.
- *Adversarially propagated transaction*: A transaction that is created by the adversary and is first seen by an honest node only through the mining of a PoW block by the adversary. Here, the adversary knows the transaction earlier than honest nodes and can get a head-start to mine it into the chains.

Our proof of the result holds regardless of how the transaction is propagated.

Probability of $V_i[r]$. In Section 3.12.3, we compute the probability, as a function of β and f, that the event $V_i[r]$ occurs for a given round r. Denote this by $\phi(\beta,f)$. Next, we find the threshold β_{thresh} such that for $\beta < \beta_{\text{thresh}}$, we have $\phi(\beta,f) > 0.5$. This means that for a transaction first revealed in round r, the probability for each chain that tx is contained in a block mined by an honest node is more than 0.5.

Achieving cross-chain-quality. We can now leverage the Chernoff bound to get a lower bound on the number of chains g, so that the probability that the adversary mined tx into more than half of the parallel chains (or equivalently, cross-chain-quality is not satisfied) is negligible in the security parameter. We show this in Section 3.12.4.

3.12.2 Honest block under $V_i[r]$

In this section, we will show that $V_i[r]$ guarantees that any transaction (honest or adversarial) first input to honest nodes at round r will be contained in an honest block in chain i.

For this, we need to show that the following hold, except with negligible probability.

- At round r, the adversary does not have a longer private adversarial chain for chain i than the longest honest chain (Lemma 26).
- The first block to be mined in chain i after round r is an honest block and this honest block persists in the longest chain for all future rounds (Lemma 27).

Lemma 26. *Consider a round when $S_i[r]=0$. The adversary cannot posses a longer chain than the chain held by the honest users at round r.*

Proof. The proof proceeds by contradiction. Suppose r was a round with $S_i[r]=0$ and the adversary possessed a longer chain C_i^a. Let the level of the tip of the longest chain held by the honest users, C_i^h, at round r be ℓ_{tip}. Let,

- $\ell_{common}=\max\{\ell : C_i^a[:\ell]=C_i^h[:\ell]\}$ be the last common ancestor,
- $\ell'=\max\{\ell \le \ell_{common}$ such that $C_i^h[\ell]$ is honest block$\}$ and r' be the round when it was mined.

Observe that for each of the blocks in the chain C_i^h from level $\ell_{common}+1$ to level ℓ_{tip}, there is a corresponding adversarial block in the adversarial chain C_i^a. Also, if $\ell'<\ell_{common}$, then all the blocks from levels $\ell'+1$ to ℓ_{common} in the chain C_i^h are adversarial blocks. Thus, from $r'+1$ to round r, the number of adversarial blocks mined is greater than the number of rounds where at least one honest block was mined. That is,

$$Z_i[r'+1,r] \ge X_i[r'+1,r]+1$$

However, we also have $S_i[r'] \ge 0 = S_i[r]$, and so,

$$\begin{aligned} 0 &\ge S_i[r]-S_i[r'] \\ &\ge \sum_{s=r'+1}^{r} Z_i[s]-X_i[s] \\ &= Z_i[r'+1,r]-X_i[r'+1,r] \end{aligned}$$

i.e. $Z_i[r'+1,r] \le X_i[r'+1,r]$ which is a contradiction. □

Lemma 27. *Suppose $V_i[r]$ happens at round r. Then, the first block B mined at round $t'>r$ is honest and will persist in the longest chain.*

Proof. First, observe that since $Z_i[r,t']=0$, we have $S_i[r-1]=S_i[r]=\cdots=S_i[t']=0$. Now, by Lemma 26, the adversary doesn't possess a longer chain in the rounds $r-1,r,\cdots,t'$. Also, $Z_i[r,t']=0$ implies that no new adversarial block was mined in in the rounds $r-1,r,\cdots,t'$ and exactly one honest node mines a block in round t'. So, the chain i held by honest users at round t' is longer than any adversarial chain and thus, contains the block B.

Next, by contradiction, let $r^*>t'$ be the earliest time when the honest users hold a chain that contains a different block at the same level where B was. Thus, at round r^*, there exists an adversarial chain that is at least as long the honest chain containing the block B. Let round r' be as defined in Lemma 26.

Then, reasoning as in Lemma 26, from round $r'+1$ to round r^*, the number of adversarial blocks mined is greater than or equal to the number of rounds where at least one honest block is mined. That is,

$$Z_i[r'+1,r^*] \geq X_i[r'+1,r^*]$$

Also, we have

$$\begin{aligned} S_i[r'] \geq 0 = S_i[r] \\ \implies Z_i[r'+1,r] \leq X_i[r'+1,r] \end{aligned}$$

This implies that $Z_i[r+1,r^*] \geq X_i[r+1,r^*]$. However, as event $V_i[r]$ occurs at round r, so, we have

$$\begin{aligned} L_i[r+1,r^*] \geq 1 \\ X_i[r+1,r^*] - Z_i[r+1,r^*] \geq 1 \\ X_i[r+1,r^*] > Z_i[r+1,r^*] \end{aligned}$$

which leads to contradiction. □

3.12.3 Computation of $\phi(f,\beta)$

In this section, we compute the probability $\phi(f,\beta)$, of the event $V_i[r]$. For this, we will prove the following lemma.

Lemma 28.

$$\lim_{f \to 0^+} \phi(f,\beta) = \frac{(1-2\beta)^2}{1-\beta}.$$

We split this computation into two parts. We compute the probability of the event $V_i^1[r]$ in Section 3.12.3, and the probability of $V_i^2[r]$ given $V_i^1[r]$ in Section 3.12.3. Finally, we can combine the two probabilities to compute $\phi(f,\beta)$.

Probability of $V_i^1[r]$

Recall that $V_i^1[r]$ is the event that $S_i[r-1] = 0$. To compute the probability of this event, we will compute the stationary distribution of the discrete time Markov chain $\{S_i[r]\}_{r \geq 0}$.

First, for a chain i, and round r, we define the following random variable:

$$\begin{aligned} S_i[r] &= (S_i[r-1] + Z_i[r] - X_i[r])_+ \\ S_i[r] &= 0 \end{aligned}$$

Observe that due to the memory-less property of both honest and adversarial mining (since mining is through a random oracle), for all $r > 0$, we have,

$$\begin{aligned} P(S_i[r] = s \,|\, S_i[r-1] = s_{r-1}, \cdots, S_i[0] = 0) \\ = P(S_i[r] = s \,|\, S_i[r-1] = s_{r-1}) \end{aligned}$$

Therefore, the process $\{S_i[r]\}_{r \geq 0}$ gives rise to a discrete time Markov chain (DTMC). We now show that $\{S_i[r]\}_{r \geq 0}$ is positive recurrent in the following lemma.

Lemma 29. *If $\mathbb{E}[Z_i[r]] < \mathbb{E}[Y_i[r]]$, then, $\{S_i[r]\}_{r \geq 0}$ is positive recurrent.*

Proof. First, notice from the definition of $S_i[r]$ that

$$\begin{aligned}(S_i[r+1])^2 &\leq (S_i[r]+Z_i[r+1]-Y_i[r+1])^2\\ &=(Z_i[r+1]-Y_i[r+1])^2\\ &\quad +(S_i[r])^2+2S_i[r](Z_i[r+1]-Y_i[r+1])\end{aligned}$$

Now, for each chain i, define a drift function as:

$$\mathfrak{d}S_i[r]=\frac{1}{2}(S_i[r+1])^2-\frac{1}{2}(S_i[r])^2$$

Then,

$$\begin{aligned}\mathfrak{d}S_i[r] &\leq \frac{1}{2}(Z_i[r+1]-Y_i[r+1])^2+S_i[r](Z_i[r+1]-X_i[r+1])\\ \mathbb{E}[\mathfrak{d}S_i[r]\,|\,S_i[r]] &\leq \left(\frac{1}{2}\mathbb{E}\big[(Z_i[r+1]-X_i[r+1])^2\,|\,S_i[r]\big]\right)\\ &\quad +(S_i[r]\mathbb{E}[(Z_i[r+1]-X_i[r+1])\,|\,S_i[r])]\end{aligned}$$

Recall that $Z_i[r+1]\sim$ Poisson(βf) and $X_i[r+1]\sim$ Bernoulli$(1-e^{-(1-\beta)f})$ and so, $Z_i[r+1]$ and $X_i[r+1]$ have bounded second moments. Therefore, the first term in the expression before can be bounded as $\frac{1}{2}\mathbb{E}\big[(Z_i[r+1]-X_i[r+1])^2\,|\,S_i[r]\big]\leq B$, where $B=B(f,\beta)$ depends on f and β. Thus,

$$\begin{aligned}\mathbb{E}[\mathfrak{d}S_i[r]\,|\,S_i[r]] &\leq B+S_i[r]\mathbb{E}[(Z_i[r+1]-X_i[r+1])\,|\,S_i[r]]\\ &=B+S_i[r](\mathbb{E}[Z_i[r+1]]-\mathbb{E}[X_i[r+1]])\end{aligned}$$

However, $\mathbb{E}[Z_i[r+1]]-\mathbb{E}[X_i[r+1]]<-\epsilon$, for some $\epsilon>0$. Therefore, we have

$$\mathbb{E}[\mathfrak{d}S_i[r]\,|\,S_i[r]]\leq B-\epsilon S_i[r]$$

To summarize,

- $\mathbb{E}[\mathfrak{d}S_i[r]\,|\,S_i[r]]\leq-\epsilon^*$ for some $\epsilon^*>0$, if $S_i[r]>\frac{B}{\epsilon}$
- $\mathbb{E}[\mathfrak{d}S_i[r]\,|\,S_i[r]]\leq B$ otherwise.

Thus, by Foster's Theorem [79], we conclude that $\{S_i[r]\}_{r\geq0}$ is positive recurrent. □

Now, $\{S_i[r]\}_{r\geq0}$ is irreducible since it is possible to get from any state to any other state (either by honest block mining or by adversarial block mining). Furthermore, it is aperiodic since it has self-loops, and therefore a 1-step transition can retain the earlier state. Therefore the stationary distribution exists. If $\boldsymbol{\pi}$ is the stationary distribution of $\{S_i[r]\}_{r\geq0}$, then, by ergodic theorem [96], we have

$$\lim_{r\to\infty}P(S_i[r]=0)=\pi_0$$

where π_0 denotes the stationary distribution probability at state 0.

We will now calculate the probability of the stationary distribution π_0 under a general f. For this, we will write the transition probability matrix for the DTMC $\{S_i[r]\}_{r\geq0}$. Define $\mathtt{p}_X^0=P(X_i[r]=0)=e^{-(1-\beta)f}$, $\mathtt{p}_X^1=P(X_i[r]=1)=1-e^{-(1-\beta)f}$, and $\mathtt{p}_Z^k=P(Z_i[r]=k)=\frac{(\beta f)^k e^{-\beta f}}{k!}$. Then, the transition

probability matrix is given by

$$\begin{pmatrix} \mathtt{p}_Z^0+\mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0+\mathtt{p}_Z^2\mathtt{p}_X^1 & \mathtt{p}_Z^2\mathtt{p}_X^0+\mathtt{p}_Z^3\mathtt{p}_X^1 & \mathtt{p}_Z^3\mathtt{p}_X^0+\mathtt{p}_Z^4\mathtt{p}_X^1 & \cdots \\ \mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0+\mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0+\mathtt{p}_Z^2\mathtt{p}_X^1 & \mathtt{p}_Z^2\mathtt{p}_X^0+\mathtt{p}_Z^3\mathtt{p}_X^1 & \cdots \\ 0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0+\mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0+\mathtt{p}_Z^2\mathtt{p}_X^1 & \cdots \\ 0 & 0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0+\mathtt{p}_Z^1\mathtt{p}_X^1 & \cdots \\ 0 & 0 & 0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The important observation is that this matrix is of the form

$$\begin{pmatrix} a & b_2 & b_3 & \cdots \\ b_0 & b_1 & b_2 & \cdots \\ 0 & b_0 & b_1 & \cdots \\ 0 & 0 & b_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

with $a=b_0+b_1$. Therefore, we get the following relationship between the stationary probabilities

$$\begin{aligned} \pi_0 &= a\pi_0+b_0\pi_1 \\ \pi_j &= \sum_{k=0}^{j+1} b_{i+1-k}\pi_k \qquad\qquad (j>1) \end{aligned}$$

Rearranging the terms of the second equation, we get

$$\begin{aligned} (1-b_1)\pi_j &= b_0\pi_{j+1}+\sum_{k=0}^{j-1} b_{j+1-k}\pi_k \\ \therefore (1-b_1)\pi_j z^{-j} &= b_0\pi_{j+1}z^{-j}+\sum_{k=0}^{j-1} b_{j+1-k}\pi_k z^{-j} \\ \therefore (1-b_1)\sum_{j\geq 1}\pi_j z^{-j} &= b_0\sum_{j\geq 1}\pi_{j+1}z^{-j}+\sum_{j\geq 1}\sum_{k=0}^{j-1} b_{j+1-k}\pi_k z^{-j} \end{aligned}$$

Notice that the summation $\sum_{j\geq 1}\sum_{k=0}^{j-1}$ can be written in terms of $\sum_{k\geq 0}\sum_{j\geq k+1}$. Now, define the unilateral z-transforms $\mathfrak{P}(z)=\sum_{j\geq 0}\pi_j z^{-j}$ and $\mathfrak{G}(z)=\sum_{j\geq 0} b_j z^{-j}$. See [96] for a primer on the z-transform. Then,

$$\begin{aligned} &(1-b_1)(\mathfrak{P}(z)-\pi_0) \\ &= b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+\sum_{k\geq 0}\sum_{j\geq k+1} b_{j+1-k}\pi_k z^{-j} = b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+z\sum_{k\geq 0}\pi_k z^{-k}\sum_{j\geq k+1} b_{j+1-k}z^{-(j+1-k)} \\ &= b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+z\mathfrak{P}(z)\left(\mathfrak{G}(z)-b_0-\frac{b_1}{z}\right) \end{aligned}$$

Rearranging, we get

$$\begin{aligned}\mathfrak{P}(z) &= \frac{\pi_0(1-b_1)-b_0 z(\pi_0+\frac{\pi_1}{z})}{1-b_0 z-z\mathfrak{G}(z)+b_0 z} \\ &= \frac{\frac{\pi_0}{z}(1-b_1)-b_0(\pi_0+\frac{\pi_1}{z})}{\frac{1}{z}-b_0-\mathfrak{G}(z)+b_0}\end{aligned}$$

Notice that the boundary condition is at $z=1$, that is, $\mathfrak{P}(1)=1$. Therefore, applying L'Hôpital's rule, we get,

$$\lim_{z\to 1} P(z) = \lim_{z\to 1} \frac{\pi_0(1-b_1)-b_0\pi_1}{1-\mathfrak{G}'(z)} = \frac{\pi_0(1-b_1)-b_0\pi_1}{1-\mathfrak{G}'(1)}$$

where $\mathfrak{G}'$ is the derivative of $\mathfrak{G}$ with respect to $\frac{1}{z}$. Consequently,

$$\begin{aligned}\pi_0(1-b_1)-(1-a)\pi_0 &= 1-\mathfrak{G}'(1) \\ \therefore \pi_0 &= \frac{1-\mathfrak{G}'(1)}{a-b_1} \\ \therefore \pi_0 &= \frac{1-\mathfrak{G}'(1)}{b_0}\end{aligned}$$

where in the last step we used the fact that $a=b_0+b_1$. Next, we will evaluate $\mathfrak{G}'(1)$. We have

$$\begin{aligned}\mathfrak{G}(z) &= \sum_{j\geq 0} b_j z^{-j} \\ \mathfrak{G}'(z) &= \sum_{j\geq 1} j b_j z^{-(j-1)}\end{aligned}$$

where we differentiated $\mathfrak{G}(z)$ with respect to $\frac{1}{z}$. Thus, we have

$$\begin{aligned}\mathfrak{G}'(1) &= \sum_{j\geq 1} j b_j \\ &= \sum_{j\geq 1} j\left[\mathtt{p}_Z^{j-1}\mathtt{p}_X^0+\mathtt{p}_Z^j\mathtt{p}_X^1\right] \\ &= \sum_{j\geq 1} j\left[\frac{(\beta f)^{j-1}e^{-\beta f}}{(j-1)!}e^{-(1-\beta)f}+\frac{(\beta f)^j e^{-\beta f}}{j!}\left[1-e^{-(1-\beta)f}\right]\right] \\ &= e^{-f}\sum_{j\geq 1} j\frac{(\beta f)^{j-1}}{(j-1)!}+\beta f e^{-\beta f}\left[1-e^{-(1-\beta)f}\right]\sum_{j\geq 1}\frac{(\beta f)^{j-1}}{(j-1)!} \\ &\overset{(a)}{=} e^{-f}\left[e^{\beta f}+\beta f e^{\beta f}\right]+\beta f e^{-\beta f}\left[1-e^{-(1-\beta)f}\right]e^{\beta f} \\ &= \beta f+e^{-(1-\beta)f}\end{aligned}$$

where in (a), we used the fact that $xe^x+x=\sum_{j\geq 1} j\frac{x^{j-1}}{(j-1)!}$ and $e^x=\sum_{j\geq 0}\frac{x^j}{j!}$. Consequently,

$$\pi_0 = \frac{1-\beta f-e^{-(1-\beta)f}}{e^{-\beta f}\left[1-e^{-(1-\beta)f}\right]}$$

Now, for $f\to 0^+$, L'Hôpital's rule can be used to evaluate π_0, which gives $\pi_0=\frac{1-2\beta}{1-\beta}$. Therefore, $\lim_{r\to\infty} P(S_i[r]=0)=\frac{1-2\beta}{1-\beta}$ as $f\to 0^+$. This means that for sufficiently small f, the probability will be sufficiently close to $\frac{1-2\beta}{1-\beta}$. Furthermore, although the stationary distribution is defined as the

round number $r \to \infty$, it will converge exponentially towards the stationary probability (see [96] for more details). This means that for a given f, at any round r after a warmup time of $T_{\text{warmup}} = \Theta(\kappa)$, the probability will be close to the stationary probability, and we can replace it with the stationary probability henceforth to simplify our computation.

Probability of $V_i^2[r]$

Next, we compute the probability of the event $V_i^2[r]$. Recall that $V_i^2[r]$ is the event as follows:

$$V_i^2[r] = \left\{ \exists t' > r \quad s.t. \quad \begin{matrix} Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \\ L_i[r+1,t] \geq 1 \quad \forall t > t' \end{matrix} \right\}$$

First, for a chain i and round $k > 0$, we define the following random walk:

$$\begin{aligned} D_i[k] &= D_i[k-1] + X_i[k] - Z_i[k] \\ D_i[0] &= 1 \end{aligned}$$

This random walk counts the difference between the number of honest blocks and number of adversarial blocks mined from round 1 to round k given that the random walk started from the initial state of $D_i[0] = 1$. Suppose that we start the random walk $\{D_i[k]\}_{k \geq 0}$ at round t'. Then,

- $Z_i[r,t'] = 0, X_i[r,t'-1] = 0$ and $Y_i[t'] = 1$ implies that two honest blocks and no adversarial blocks were mined from the round r to round t'. Therefore, we set $D_i[0] = 1$.

- For any $k > 0$, $D_i[k] = L[r+1, t'+k]$.

Now, $V_i^2[r] = \{D_i[0] = 1, D_i[k] \geq 1 \quad \forall k > 0\}$, and therefore, we get:

$$\begin{aligned} P(V_i^2[r]) &= \{D_i[0] = 1, D_i[k] \geq 1 \quad \forall k > 0\} \\ &= P(D_i[0] = 1) \times P(D_i[k] \geq 1 \quad \forall k > 0 \,|\, D_i[0] = 1) \end{aligned}$$

First, we compute $P(D_i[0] = 1)$ as

$$\begin{aligned} P(D_i[0] = 1) &= P\begin{pmatrix} \exists t' > r \text{ such that} \\ Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \end{pmatrix} \\ &= \sum_{m=1}^{\infty} P\begin{pmatrix} t' = r+m \\ Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \end{pmatrix} \\ &= \sum_{m=1}^{\infty} \left(e^{-(1-\beta)f}\right)^m \left(e^{-\beta f}\right)^{m+1} (1-\beta) f e^{-(1-\beta)f} \\ &= \frac{(1-\beta) f e^{-2f}}{1 - e^{-f}} \end{aligned}$$

Now, we can use L'Hôpital's rule, to get $\lim_{f \to 0^+} P(D_i[0] = 1) = 1 - \beta$.

In order to compute $P(D_i[k] \geq 1 \quad \forall k > 0 \,|\, D_i[0] = 1)$, we first define

$$\mathbf{P}_u = P(D_i[r] \geq 1 \quad \forall r \geq 0 \,|\, D_i[0] = u)$$

Observe that the boundary conditions are $\mathbf{P}_0 = 0$ and $\mathbf{P}_\infty = 1$. Now for $u \geq 1$,

$$
\begin{aligned}
\mathbf{P}_n &= P(D_i[r] \geq 1 \quad \forall r \geq 0 \,|\, D_i[0] = u) \\
&= \sum_{k=-1}^{\infty} P(X_i[1] - Z_i[1] = -k, D_i[r] \geq 1 \quad \forall r \geq 0 \,|\, D_i[0] = u) \\
&= \sum_{k=-1}^{\infty} P(X_i[1] - Z_i[1] = -k) P(D_i[r] \geq 1 \quad \forall r \geq 0 \,|\, D_i[1] = u-k) \\
&= \sum_{k=-1}^{u-1} P(X_i[1] - Z_i[1] = -k) P(D_i[r] \geq 1 \quad \forall r \geq 0 \,|\, D_i[1] = u-k) \\
&= \sum_{k=-1}^{u-1} P(X_i[1] - Z_i[1] = -k) \mathbf{P}_{u-k} \\
&= a\mathbf{P}_{u+1} + c\mathbf{P}_u + \sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k}
\end{aligned}
$$

where $a = P(X_i[1] - Z_i[1] = 1)$, $c = P(X_i[1] - Z_i[1] = 0)$ and $b_k = P(X_i[1] - Z_i[1] = -k)$. Therefore,

$$
\begin{aligned}
(1-c)\mathbf{P}_u z^{-u} &= a\mathbf{P}_{u+1} z^{-u} + \sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k} z^{-u} \\
(1-c)\sum_{u \geq 1} \mathbf{P}_u z^{-u} &= az \sum_{u \geq 1} \mathbf{P}_{u+1} z^{-(u+1)} + \sum_{u \geq 1} \sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k} z^{-u}
\end{aligned}
$$

We now define the z-transforms $\mathfrak{P}(z) = \sum_{u \geq 1} \mathbf{P}_u z^{-u}$ and $\mathfrak{B}(z) = \sum_{k \geq 0} b_{k+1} z^{-k}$. We can skip $\mathbf{P}_0$ as $\mathbf{P}_0 = 0$. Therefore,

$$
\begin{aligned}
(1-c)\mathfrak{P}(z) &= az(\mathfrak{P}(z) - \mathbf{P}_1 z^{-1}) + \sum_{k \geq 1} \sum_{u \geq k+1} b_k \mathbf{P}_{u-k} z^{-k} z^{-(u-k)} \\
&= a(z\mathfrak{P}(z) - \mathbf{P}_1) + \frac{1}{z} \sum_{k \geq 1} b_k z^{-(k-1)} \sum_{u \geq k+1} P_{u-k} z^{-(u-k)} \\
&= a(z\mathfrak{P}(z) - \mathbf{P}_1) + \frac{1}{z} \mathfrak{B}(z) \mathfrak{P}(z) \\
\therefore \mathfrak{P}(z) &= \frac{a\mathbf{P}_1 z}{az^2 - (1-c)z + \mathfrak{B}(z)}
\end{aligned}
$$

Recall that,

- $a = P(X_i[1] - Z_i[1] = 1) = P(X_i[1] = 1) P(Z_i[1] = 0) = (1 - e^{-(1-\beta)f}) e^{-\beta f} = e^{-\beta f} - e^{-f}$
- $c = P(X_i[1] - Z_i[1] = 0)$
  $= P(X_i[1] = 0) P(Z_i[1] = 0) + P(X_i[1] = 1) P(Z_i[1] = 1)$
  $= e^{-(1-\beta)f} e^{-\beta f} + (1 - e^{-(1-\beta)f}) \beta f e^{-\beta f}$
  $= e^{-f} + \beta f (e^{-\beta f} - e^{-f})$
- $b_k = P(X_i[r] - Z_i[r] = -k)$
  $= P(X_i[r] = 0) P(Z_i[r] = k) + P(X_i[r] = 1) P(Z_i[r] = k+1)$
  $= e^{-(1-\beta)f} \frac{(\beta f)^k e^{-\beta f}}{k!} + (1 - e^{-(1-\beta)f}) \frac{(\beta f)^{k+1} e^{-\beta f}}{(k+1)!}$
  $= e^{-f} \frac{(\beta f)^k}{k!} + (e^{-\beta f} - e^{-f}) \frac{(\beta f)^{k+1}}{(k+1)!}$

Therefore, $\lim_{f\to 0^+}\mathfrak{B}(z)=0$. Now, by L'Hôpital's rule,

$$\begin{aligned}\lim_{f\to 0^+}\mathfrak{P}(z)&=\frac{(1-\beta)\mathbf{P}_1 z}{(1-\beta)z^2-z+\beta}\\&=\frac{(1-\beta)\mathbf{P}_1}{\beta}\left[\left(\frac{1-\beta}{1-2\beta}\right)\left(\frac{1}{z^{-1}-\frac{1-\beta}{\beta}}\right)-\left(\frac{\beta}{1-2\beta}\right)\left(\frac{1}{z^{-1}-1}\right)\right]\\&=\frac{(1-\beta)\mathbf{P}_1}{1-2\beta}\left[\left(\frac{1-\beta}{\beta}\right)\left(\frac{1}{z^{-1}-\frac{1-\beta}{\beta}}\right)-\left(\frac{1}{z^{-1}-1}\right)\right]\end{aligned}\tag{3.1}$$

To determine $\mathbf{P}_u$, we need to perform an inverse z-transform of $\mathfrak{P}(z)$ as shown below:

$$\mathbf{P}_u=\frac{1}{2\pi j}\oint_C\mathfrak{P}(z)z^{u-1}dz$$

where j is the imaginary unit, and C is a counterclockwise closed path encircling the origin and entirely lying in the region of convergence. Next, we want to determine $\lim_{f\to 0^+}\lim_{n\to\infty}\mathbf{P}_u$. However, we have,

$$\lim_{f\to 0^+}\lim_{u\to\infty}\mathbf{P}_u=\lim_{u\to\infty}\lim_{f\to 0^+}\mathbf{P}_u$$

wherein we used Moore-Osgood theorem [150] because:

- $\lim_{n\to\infty}\mathbf{P}_u$ exists pointwise for each different $f\neq 0$ because, by definition, $0\leq\mathbf{P}_u\leq 1$ for all $u\geq 0$
- $\lim_{f\to 0^+}\mathbf{P}_u$ converges uniformly. This is because,
  1. The integral in the inverse z-transform is over a contour C that lies in the region of convergence and so $\mathfrak{P}(z)z^{u-1}$ is dominated by some Lebesgue integral function $\mathfrak{S}(z)$ in the sense that $|\mathfrak{P}(z)z^{u-1}|\leq\mathfrak{S}(z)$.
  2. Therefore, by the dominated convergence theorem [150] and using equation 3.1, we can interchange the limit and the contour integral to obtain

$$\begin{aligned}\lim_{f\to 0^+}\mathbf{P}_u&=\lim_{f\to 0^+}\frac{1}{2\pi j}\oint_C\mathfrak{P}(z)z^{u-1}dz=\frac{1}{2\pi j}\oint_C\lim_{f\to 0^+}\mathfrak{P}(z)z^{u-1}dz\\&=\mathbf{P}_1\left(\frac{1-\beta}{1-2\beta}\right)\left[1-\left(\frac{\beta}{1-\beta}\right)^u\right].\end{aligned}$$

  Further, note that $0\leq\mathbf{P}_1\leq 1$.
  3. Observe that $\mathbf{P}_1\left(\frac{1-\beta}{1-2\beta}\right)\left[1-\left(\frac{\beta}{1-\beta}\right)^u\right]$ uniformly converges to $\mathbf{P}_1\left(\frac{1-\beta}{1-2\beta}\right)$.

Recall now, that our boundary condition was $P_\infty=1$ i.e.

$$1=\lim_{u\to\infty}\mathbf{P}_u=\lim_{f\to 0^+}\lim_{u\to\infty}\mathbf{P}_u=\mathbf{P}_1\left(\frac{1-\beta}{1-2\beta}\right)$$

Finally, we get

$$\mathbf{P}_1=\frac{1-2\beta}{1-\beta}$$

i.e., $\lim_{f\to 0^+}P(D_i[k]\geq 1\quad\forall k>0\,|\,D_i[0]=1)=\frac{1-2\beta}{1-\beta}$. Now, we can compute $P(V_i^2[r])$ as,

$$\begin{aligned}\lim_{f\to 0^+}P(V_i^2[r])&=\frac{1-2\beta}{1-\beta}\left(\lim_{f\to 0^+}P(D_i[0]=1)\right)\\&=\frac{1-2\beta}{1-\beta}(1-\beta)=(1-2\beta)\end{aligned}$$

Final computation of $\phi(f,\beta)$

Having computed the probabilities of $V_i^1[r]$ and $V_i^2[r]$, we can now complete the computation of $\phi(f,\beta)$ as,

$$\begin{aligned}\lim_{f\to 0^+}\phi(f,\beta) &= \lim_{f\to 0^+}P(V_i[r]) = \lim_{f\to 0^+}P(V_i^1[r])\lim_{f\to 0^+}P(V_i^2[r])\\ &= \frac{(1-2\beta)}{1-\beta}(1-2\beta) = \frac{(1-2\beta)^2}{1-\beta}\end{aligned}$$

This completes the proof of Lemma 28. Now, since we want to bound β such that $\phi(f,\beta)>0.5$ (for a sufficiently small f), we get $8\beta^2-7\beta+1>0$. Recall that the Nakamoto consensus properties are not applicable for $\beta>\frac{1}{2}$. Therefore, we end up with the bound $\beta<\frac{7-\sqrt{17}}{16}\approx 0.1798$.

For our (τ,R)-respawning adversaries, recall that another τ fraction of nodes can get killed by the time a transaction is included in all parallel chains, and therefore, we will require $\beta+\tau<\frac{7-\sqrt{17}}{16}$. We also node that this bound may not be tight and that a larger adversarial threshold might still be acceptable.

3.12.4 Achieving cross-chain-quality

Equipped with the computation of $\phi(f,\beta)=P(V_i[r])$ from the previous section, we can proceed to calculate a bound on the number of chains d. For this, we first declare an event that needs to happen with negligible probability.

An unlikely event. Consider the following event T for any $\epsilon>0$.

$$T := \left\{\exists r\in\{1,\cdots,|\mathsf{view}|\} : \frac{1}{d}\sum_{i=1}^{d}\mathbf{1}_{V_i[r]} < \frac{1}{2}-\epsilon\right\}$$

The event T implies that if any transaction is revealed to the public in round r, then in at most $\frac{1}{2}-\epsilon$ fraction of the chains, an honest block will contain the transaction. In the following lemma, we show that if $\phi(f,\beta)>\frac{1}{2}$, then for sufficiently large d, $P(T)$ will be negligible in κ.

Lemma 30. *Suppose β and a sufficiently small constant mining rate f are appropriately chosen such that $\phi(f,\beta)>\frac{1}{2}$. Then, $P(T)\leq|\mathsf{maxview}|e^{-\Omega(d)}$ where* maxview *is the longest view.*

Proof. Using the union bound,

$$\begin{aligned}P(T) &= P\left(\exists r\in\{1,\cdots,|\mathsf{view}|\} : \frac{1}{d}\sum_{i=1}^{d}\mathbf{1}_{V_i[r]} < \frac{1}{2}-\epsilon\right)\\ &\leq \sum_{1\leq r\leq|\mathsf{maxview}|} P\left(\frac{1}{d}\sum_{i=1}^{d}\mathbf{1}_{V_i[r]} < \frac{1}{2}-\epsilon\right)\\ &= \sum_{1\leq r\leq|\mathsf{maxview}|} P\left(\sum_{i=1}^{d}\mathbf{1}_{(V_i[r])^C} > \left(\frac{1}{2}+\epsilon\right)d\right) \qquad (3.2)\end{aligned}$$

where $(V_i[r])^C$ is the complement of $V_i[r]$. Note that $P(\mathbf{1}_{(V_i[r]^C)})=1-\phi(f,\beta)$. Since the mining for all chains is independent, we can consider $\mathbf{1}_{(V_1[r])^C},\cdots,\mathbf{1}_{(V_g[r])^C}$ to be independent of each other. As

$1-\phi(f,\beta) < \frac{1}{2}$, therefore, for some $\delta > 0$, we can write

$$(1+\delta)d(1-\phi(f,\beta)) = \left(\frac{1}{2}+\epsilon\right)d$$
$$\delta = \frac{\left(\frac{1}{2}+\epsilon\right)}{1-\phi(f,\beta)} - 1$$

Then, using the Chernoff bound on equation 3.2, we have

$$\begin{aligned} P(T) &\leq \sum_{1 \leq r \leq |\mathsf{maxview}|} e^{-\frac{\delta^2 d[1-\phi(f,\beta)]}{3}} \\ &\leq \sum_{1 \leq r \leq |\mathsf{maxview}|} e^{-\Omega(d)} \\ &= |\mathsf{maxview}| e^{-\Omega(d)} \end{aligned}$$

□

Now, note that since $|\mathsf{maxview}|$ is polynomial in κ, setting $d = \Theta(\kappa)$ results in $P(T)$ being negligible in κ.

Chapter 4

Themis : Fast, Strong Order-Fairness in Byzantine Consensus

We introduce Themis , a scheme for introducing *fair ordering* of transactions into (permissioned) Byzantine consensus protocols with at most f faulty nodes among $n \geq 4f+1$. Themis enforces the strongest notion of fair ordering proposed to date. It also achieves standard liveness, rather than the weaker notion of previous work with the same fair ordering property.

We show experimentally that Themis can be integrated into state-of-the-art consensus protocols with minimal modification or performance overhead. Additionally, we introduce a suite of experiments of general interest for evaluating the practical strength of various notions of fair ordering and the resilience of fair-ordering protocols to adversarial manipulation. We use this suite of experiments to show that the notion of fair ordering enforced by Themis is significantly stronger in practice than those of competing systems.

We believe Themis offers strong practical protection against many types of transaction-ordering attacks—such as front-running and back-running—that are currently impacting commonly used smart contract systems.

4.1 Introduction

Decentralized Finance (DeFi), meaning the deployment of financial instruments on blockchains, has attracted substantial interest in recent years, with over 45 billion USD locked in DeFi protocols as of Jan 2023 [16]. Unfortunately, while DeFi continues to gain popularity, a long line of work [63, 74, 108, 170, 140] has shown the rise of adversaries extracting profit by manipulating the ordering and inclusion of transactions in DeFi applications. In decentralized exchanges and lending contracts, for example, where transaction execution order is critically important, such *order manipulation* results in attackers profiting at the expense of ordinary users.

Order manipulation is possible in existing protocols largely because the formal properties required of state machine replication (SMR) or consensus—the primitive that underpins blockchains—place *no restriction on how transactions are ordered*. Neither consistency nor liveness, the two pillars of consensus security, enforces any relationship between the order in which transactions arrive in the network and their final ordering. Indeed, in both permissioned consensus protocols, e.g., PBFT [49] and Hotstuff [163], and permissionless ones, e.g., Ethereum, the current "leader" fully controls the inclusion and ordering of transactions within a block that it creates.

To address this gap in traditional consensus research, a recent line of work [105, 102, 168, 110, 111, 44] has proposed protocols with so-called *fair ordering* properties—properties that prevent adversarial manipulation of transaction ordering. These works propose several definitions of *fairness*[1] along with protocols that realize them. Intuitively, this style of fairness seeks to guarantee a specific ordering in

[1] We use "fairness" to mean *fairness of transaction ordering* or *fair ordering*, although the term has been used in the past for unrelated notions (e.g., fair PoW mining [133]).

the finalized ledger based on how transactions arrive into the network. These notions are different and in many cases stronger than past ordering properties such as causal ordering [144, 42] which only prevents reordering of transactions based purely on their content and fails to account for a range of attacks, e.g., those based on metadata leakage or prioritizing adversarial transactions over others (e.g., to get the best purchase price for an asset [122]). The new line of work on fair ordering attempts to tackle ordering at a more fundamental level; notably, [105, 102, 168] all found exciting connections of the fair ordering problem to social choice theory.

The fair ordering landscape. Existing fair-ordering protocols, however, have serious practical limitations. The Aequitas protocol from [105] has impractically high $\mathcal{O}(n^3)$ communication complexity and is also only able to provide a weaker liveness property. Protocols from subsequent works require $\mathcal{O}(n^2)$ communication but suffer from significant other shortcomings.

The protocol in [44] (concurrent to our work) is only shown to provide liveness when *all nodes are honest* (see Section 4.6). Moreover, as we show in this work (see Section 4.5), there are subtle *censorship issues* in `Pompē` [168] and that the fairness property satisfied by both `Pompē` [168] and `Wendy` [110] is *significantly weaker* than one from Aequtias. Table 5.1 shows a few comparison points.

This work presents a new protocol `Themis` , which we term to be the first fair-ordering protocol that can be practically deployed. `Themis` achieves the same strong fairness property as Aequitas, guarantees liveness, and our implementation incurs minimal cost over Hotstuff [163], a widely used state-of-the-art consensus protocol without any fair-ordering guarantees.

4.1.1 `Themis` Overview and Contributions

`Themis` operates in a partially synchronous setting with a committee of n nodes of which at most f may be arbitrarily adversarial where $n \geq 4f+1$. We implement `Themis` on top of Hotstuff [163], a widely used leader-based protocol and show that it incurs minimal performance overhead.

`Themis` : Fair-ordering property. `Themis` achieves the *batch-order-fairness* property proposed by Kelkar et al. [105]. Informally, batch-order-fairness [2] (Definition 18) with parameter $\frac{1}{2} < \gamma \leq 1$ dictates that if γ fraction of nodes receive a transaction tx before tx' from the client, then tx should be ordered *no later than* tx'. While [105] also introduced a stronger receive-order-fairness property where tx must be ordered strictly before tx', it was shown to be impossible without strong synchrony assumptions.

Informally, *batches* arise as a result of non-transitive *Condorcet* cycles [8] in message receipt times across nodes (see Section **??** for more details); the relaxation is minimal in that when there are no cycles, the stronger receive-order-fairness property can be satisfied.

We find however that these cycles can extend for arbitrarily long; this is ultimately responsible for the liveness problems with Aequitas. To get around this, we notice that in a typical deployment for `Themis` (e.g., for smart contracts), all transactions (even those in a batch) will require a total ordering for execution. Here, unlike Aequitas which can totally order transactions in a cycle only after all transactions in it are seen (which can take arbitrarily long), in `Themis` , using our technique of *unspooling* (see Section **??**), we can output a batch part-by-part while still ensuring that all transactions in the same batch are output in an uninterrupted sequence. As a consequence, `Themis` imposes a total ordering on transactions, but one that respects batch-order-fairness .

`Themis` further guarantees another useful property: for any two consequent transactions tx_j and tx_{j+1} in the final output, it holds that tx_j was received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes. This property also provides resistance against a particular kind of frontrunning attack—where the adversary wants to *immediately precede* a targeted user transaction.

We find that the fairness property supported in `Themis` (even for the weakest version, i.e., with $\gamma = 1$) is stronger in practice than other notions proposed in Zhang et al. [168] and Kursawe [110]. This is showcased through our suite of fairness experiments.

[2] Here, *batch* is unrelated to the standard SMR optimization of increasing throughput (at the cost of latency) by amortizing consensus over many transactions.

`Themis` **design** (Section 4.3). `Themis` can be bootstrapped from any leader-based consensus protocol with minimal design changes: First, before constructing a block, all replicas send information about the order in which they received client transactions to the current leader. Second, we specify an algorithm for an honest leader to construct a fair block proposal from the replica orderings. Finally, we provide a way for the replicas to verify the fairness of a leader's proposal as well as extract out the final ordering.

To construct a fair proposal, we extract out key techniques from [105] for ordering transactions (much as [102] does). Unfortunately, applying these techniques naïvely results in a loss of liveness, similar to the Aequitas protocol from [105] which achieves only a weaker liveness notion due to the possible "chaining" of Condorcet cycles (see Section **??**). Concretely, in `Themis` , loss of liveness would mean empty blocks being produced by several honest leaders until the "chaining" of the current Condorcet cycle is completed.

Our solution is a new technique that we call *deferred ordering*. With deferred ordering, blocks produced by a leader contain some transactions that are fully ordered, while other transactions are only partially ordered. Partially ordered transactions await total ordering by a subsequent honest leader. Notably, the finalization of these partially ordered transactions *happens within the network delay* and does not have to wait indefinitely for the ordering of future transactions, e.g., the presence of Condorcet cycles. This feature of `Themis` allows us to circumvent the liveness problem of Aequitas. Thanks to deferred ordering, `Themis` achieves the standard liveness property. The technique is also of general interest: it can, e.g., be retrofitted to [105] to achieve standard liveness there too.

Theoretical SNARK-based design (Section 4.3.3 and Appendix 4.8.3). We also describe a more theoretical design—SNARK-Themis —which makes use SNARKs to achieve optimistic $\mathcal{O}(n)$ communication complexity. This is particularly notable since we find that other fair-ordering protocols cannot easily be made optimistically $\mathcal{O}(n)$ even with the use of SNARKs. This makes SNARK-Themis the first such fair-ordering protocol; it's asymptotic communication complexity is in fact optimal and equivalent to state-of-the-art protocols that lack any fair ordering guarantees, e.g., Hotstuff [163].

Implementation and benchmarks (Section 4.4). We implement the $\mathcal{O}(n^2)$ version of `Themis` , and show that its integration with Hotstuff's codebase [1] results in small performance overhead in practice; our implementation with $n = 30$ nodes was still able to achieve a latency of roughly 53ms and a peak throughput of 52,719 transactions per second, which should suffice for most applications. Notably, `Themis` scales exactly as Hotstuff as n is increased. Furthermore, any overhead almost entirely vanishes when nodes are geo-distributed. `Themis` ' source code is available at: `https://github.com/anonthemis/themis-src-anon`github.com/anonthemis/themis-src-anon. Since the public release of our paper, a prototype implementation of Themis has also been independently developed by Chainlink (the largest provider of oracle services for blockchain applications) as a practical emerging solution to the problem of order-manipulation [51].

Suite of fairness experiments (Section 4.5). There exists no prior work on practical measurement or empirical comparison of fair-ordering protocols. A key contribution of our work, therefore, is a systematically conceived suite of experiments to quantify the practical impact of both fairness definitions and protocols and understand their design tradeoffs. We consider both honest settings as well as broad classes of adversarial attacks common in practice. We study `Themis` through this lens and show that it provides significantly better fairness properties compared to other alternatives. We believe that our fairness suite will be useful for any future work on fair ordering protocols. We showcase three experiments:

1. *Ideal setting.* We quantify the strength of different fair ordering properties in an ideal honest setting (with no adversarial nodes) to understand the best-case scenario.

2. *Frontrunning and insertion attacks.* We evaluate resilience against network-layer insertions—attempts to maliciously insert transactions at the network layer (i.e., even before the consensus begins) through e.g., frontrunning.

We first prove that `Themis` does not allow any frontrunning under a natural assumption that the network respects triangle inequality. This result complements our experiments using a real network. As a concrete datapoint, for 100 geo-distributed nodes, for the fairness notions in [168, 110], up to 94% of the node connections in the network are still susceptible to adversarial frontrunning. In contrast, for the fairness notion in `Themis` , the number goes down to 2.8% in the simplest protocol parameter choice, and just 0.16% for the optimal parameter.

3. *Adversarial reordering.* We evaluate robustness to reordering attacks, i.e., attempts to maliciously reorder transactions compared to the honest execution. As a concrete datapoint, in setting with 101 nodes, it is easier to reorder in a median-timestamp based protocol (which essentially abstracts out the fairness components of [168, 110]) with just 5 adversarial nodes, than in `Themis` even given 25 nodes.

As a separate point, we also evaluate resistance to *censorship* (Section 4.5.6). We show a subtle censorship issue with `Pompē` [168], due to its use of an ordering phase prior to the actual consensus protocol.

4.2 Preliminaries

Model. Our setup is a permissioned system with a set $\mathcal{N}$ of n known protocol *nodes* or *replicas*, of which at most f are controlled by an adversary (denoted by $\mathcal{A}$) and can deviate arbitrarily from the protocol description. Transactions to be sequenced are sent by system clients to all replicas. For our fair ordering protocols, we will consider the times at which a transaction was received by the replicas to decide on its overall ordering in the final ledger. For communication between replicas, we assume the presence of a PKI, and the security of digital signatures. The network itself is partially synchronous [72]; specifically, there exists a network delay Δ that bounds the message delivery time between replicas, but is not known to the replicas. $\mathcal{A}$ controls all message delivery, and can delay and reorder messages up to the bound Δ.

Graph terminology and algorithms. We make use of common graph algorithms to reason about the ordering dependencies between transactions. Let $\mathcal{G}=(\mathcal{V},\mathcal{E})$ denote a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Unless specified, all graphs will be directed and unweighted. We often use vertices and transactions interchangeably when referring to graphs where vertices represent transactions.

$\mathcal{G}$ is a *tournament* graph if there is exactly one edge between each pair of vertices. For $V \in \mathcal{V}$, $\mathsf{SCC}_{\mathcal{G}}(V)$ denotes the strongly connected component that contains V. The subscript can be dropped when the context is clear. Recall that an SCC is a maximal subgraph such that there is a path in each direction between each pair of vertices in the component. $\mathcal{G}^*$ denotes the condensation of $\mathcal{G}$ (i.e., the transformed graph where vertices in the same SCC are collapsed into a single vertex). Note that $\mathcal{G}^*$ is guaranteed to be acyclic.

Within a graph, a Hamiltonian path is a path (i.e., a sequence of vertices) that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path that forms a cycle, i.e., there is also an edge from the last vertex to the first vertex in the path. For an acyclic graph $\mathcal{G}$, a topological sorting is a linear ordering of vertices such that for any vertices U and V, U is ordered before V if $(U,V) \in \mathcal{G}.\mathcal{E}$.

Given a graph $\mathcal{G}$, its condensation can be easily computed in time $\mathcal{O}(|\mathcal{V}|+|\mathcal{E}|)$ using depth-first search techniques [157]. Moreover, if $\mathcal{G}$ is acyclic, then it can also be topologically sorted with the same asymptotic complexity. While the problem of detecting Hamiltonian paths in generic graphs is NP-complete, for acyclic graphs in particular, it is solvable in time $\mathcal{O}(|\mathcal{V}|+|\mathcal{E}|)$ (through topological sorting). This is also true for tournament graphs [121]. Our starting point is the fairness definitions of Kelkar et al. [105], and their Aequitas fair-ordering protocol. Our experiments show that their fairness property is stronger than the ones from other works, and therefore the focus of this work is to achieve the same strong property while fixing two problems with Aequitas: (1) the lack of standard liveness; and (2) the high communication complexity.

4.2.1 Aequitas Background

Batch-order-fairness. While Kelkar et al. [105] defined several fairness variants, the primary notion considered and subsequently realized by their leaderless Aequitas protocol, was batch-order-fairness , parameterized by $\frac{1}{2} < \gamma \leq 1$.

Definition 18 (γ-batch-order-fairness)**.** *Suppose that* tx *and* tx′ *are received by all nodes. If* γn *nodes received* tx *before* tx′ *locally, then all honest nodes output* tx *no later than* tx′*.*

A stronger notion, receive-order-fairness (exactly as Definition 18, but now, tx must be output *before* tx′), was also considered. An impossibility result, however, rules out realization except in specific synchronous settings. The impossibility arises from the Condorcet paradox [8] in voting theory.

Abstractly, this allows for non-transitive global preferences even if each party's local preferences are transitive. As a simple example, suppose that three nodes receive transactions in the order $[a,b,c]$,$[b,c,a]$ and $[c,a,b]$. Here, each of "a before b", "b before c", and "c before a" holds for a majority of nodes, resulting in a non-transitive global preference.

Consequently, the global "fair" transaction ordering (according to receive-order-fairness) could contain cycles (even in a non-adversarial setting), which we call *Condorcet cycles* (we will often simply use the term cycle). Batch-order-fairness sidesteps this problem by enabling transactions to be output in *batches* (transactions within a batch can still be totally ordered at e.g., the application layer), and subsequently ignoring unfairness resulting from any cyclic orderings in the same batch. Importantly, **the batch relaxation is only used for transactions in the same cycle**.

Aequitas overview. The Aequitas protocol from [105] consists of three stages that each transaction goes through before being delivered to the ledger. First, in the *gossip* stage, all nodes broadcast transactions in the order they were locally received from clients. FiFo broadcast [97] is used to ensure that the broadcast order of an honest sender is maintained. Next, in the *agreement* stage, a variant of Byzantine Agreement [112] is used to agree on whose local orderings to use to order a particular transaction. Lastly, the *finalization* stage is used to non-interactively determine the final transaction ordering. Given the local orderings of other nodes, the finalization algorithm builds a dependency graph of transactions as they arrive. Here, edges represent ordering dependencies between transactions (e.g., an edge from tx to tx′ signifies that a large number of nodes have received tx earlier). Disregarding the complexity introduced by the graph being built at different rates by different nodes, the core algorithm removes transactions from the graph and outputs them when, informally, they no longer have any dependencies.

Aequitas realizes batch-order-fairness and circumvents the receive-order-fairness impossibility by delivering transactions in the same Condorcet cycle at the same time (i.e., in the same "batch"). Further, Aequitas guarantees that batch-relaxation is minimal—if there are no Condorcet cycles, each batch includes only a single transaction (i.e., the stronger receive-order-fairness notion will be met).

4.2.2 Aequitas Technical Challenges

Weak-liveness. A crucial problem with the Aequitas protocol, however, is that it was proven to only guarantee a *weak* notion of liveness (given as Definition **??** for completeness). We find that once again, the presence of Condorcet cycles proves problematic here. In particular, we find that Condorcet cycles can "chain" together and form larger ones that can extend for arbitrarily long (in the worst case); informally, two cycles can be chained by having them share a transaction. This means that transactions input much later can become part of the same cycle (although this can happen because of the specific input transaction orderings and not necessarily the adversary). In other words, specific input orderings could prevent transactions from being output for arbitrarily long.

Although Kelkar et al. [105] do not formally define these cycles, they are implicit in their weak-liveness definition. We show that, intuitively, an equivalent formulation of weak-liveness is that liveness for a transaction is guaranteed *only after its entire cycle is complete*, which may take arbitrarily long in the worst case (i.e., it cannot be a function of Δ). In contrast, (standard) liveness in a partially synchronous network guarantees that transactions are output within a finite time dependent on the fixed (but unknown) parameter Δ.

Technical challenges. Fixing the two aforementioned problems (weak-liveness and communication complexity) with Aequitas results in some technical challenges that required novel insights. Surprisingly, we find that the two problems are intertwined in a way that solving one actually requires solving the other.

We first prove that since Condorcet cycles can extend for arbitrarily long, if the protocol waits for all transactions in a cycle to be seen before they are output, it cannot achieve standard liveness; in fact, here, Aequitas' weak-liveness is the best possible property.

To get around this, we notice that in a standard SMR deployment (e.g., for smart contracts), all transactions (even those in the same batch) will need to be *totally ordered* for execution. Note that this is compatible with batch-order-fairness and here, Aequitas will enforce any total ordering within a batch after it has been *fully seen*. Surprisingly, it turns out that designing the protocol from the ground up with the final total ordering in mind allows us to get around the liveness problems. In so doing, we introduce *batch unspooling*, which intuitively allows for a batch to be output *part-by-part* (i.e., some transactions in a batch output before the rest) while still ensuring that all transactions in the batch are output *contiguously* (i.e., all transactions in a batch are output in an uninterrupted sequence). Towards this goal, our work needed to establish a **novel understanding of Condorcet cycles**.

Second, to reduce the communication cost, we use a common technique of routing communication through a *leader*. This turns out to be quite tricky however; a naïve design actually leads to higher cost. This happens because the presence of arbitrary-length Condorcet cycles can result in empty proposals even from honest leaders. To solve this, we introduce our technique of *deferred ordering* which enables the unspooling of a batch *across leaders*. In particular, deferred ordering allows for a leader proposal, in some cases, to contain partially ordered transactions which will be be totally ordered by a subsequent leader. Notably, we can still guarantee that the total ordering will be finalized *only based on the network delay* and therefore no weak-liveness problems arise.

4.2.3 Novel Understanding of Condorcet Cycles

While understanding cyclic ordering dependencies between transactions, we need to account for the fact that some replicas may be adversarial (and claim to have received transactions in a different order), as well as the fact that up to f honest nodes may not be considered (since we need to work in partial synchrony).

Condorcet cycles. We formally define (weak)-Condorcet cycles below as they lead to a key piece of our protocol design; these turn out to also be hidden within the weak-liveness definition of [105].

Definition 19 (Condorcet Cycle)**.** *A list* $[\mathrm{tx}_1,\ldots,\mathrm{tx}_l]$ *is a (weak)-Condorcet cycle of length* l *if the following holds: For all* $i \in \{1,\ldots,l\}$*, where* $\mathrm{tx}_1 = \mathrm{tx}_{l+1}$*, at least* $n(1-\gamma)+1$ ***honest*** *nodes have received* tx_{i+1} *before* tx_i*.*

Remark 9. *While not important for our work, we can also define a stronger version based on the transaction ordering nodes "claim" to have received. In particular, in the above definition, for each* i*, if at least* $\gamma n - f$ *nodes "claim" to have received* tx_{i+1} *before* tx_i *(honest nodes claim the order in which they receive transactions while adversarial nodes can claim arbitrary orderings), then* $[\mathrm{tx}_1,\ldots,\mathrm{tx}_l]$ *will become a strong-Condorcet cycle. The threshold* $\gamma n - f$ *is used here since it is guaranteed to hold when* γn *nodes receive* tx_{i+1} *before* tx_i*. Unless specified, we will use cycles from Definition 19.*

Impossibility of liveness for entire batches. We start by showing a constructive example of an input transaction ordering that results in Condorcet cycles of *arbitrary length (and time)* for any parameters n, $f \geq 1$, and γ. This is done by continuously *chaining* together smaller cycles to form larger ones. As an informal but illustrative example, a cycle $[\mathrm{tx}_1,\mathrm{tx}_2,\ldots,\mathrm{tx}_l]$ can be formed through smaller cycles $[\mathrm{tx}_1,\mathrm{tx}_2,\mathrm{tx}_3]$, $[\mathrm{tx}_3,\mathrm{tx}_4,\mathrm{tx}_5]$ and so on where subsequent cycles are formed such that they share a common transaction. The overall construction is somewhat non-intuitive, but we give a general algorithm in Appendix **??** to explicitly build these arbitrary length cycles. In fact, we also show something stronger, namely the construction of arbitrary-length *strong*-Condorcet cycles.

As a consequence, if a protocol waits for the entire cycle to be seen before transactions within it are output (as Aequitas does), then standard liveness is impossible to achieve.

Batch *unspooling.* We bypass this impossibility using a new technique. In typical consensus use cases (e.g., smart-contract settings), it is necessary to *totally order* all transactions for execution. For this, Aequitas supports enforcing any total ordering for transactions within a batch. Still, this is done only at the end *after* the entire cycle has been seen. The final ordering can now be thought of being linearly partitionable into cycles; in particular, $[\text{tx}_1^{(1)},...,\text{tx}_{l_1}^{(1)},\text{tx}_1^{(2)},...,\text{tx}_{l_2}^{(2)},...]$ where $\text{tx}_1^{(i)},...,\text{tx}_{l_i}^{(i)}$ are part of the same cycle.

Perhaps surprisingly, we find that embedding the total ordering requirement *within the protocol design* itself allows us to provide (standard) liveness while still *keeping the same fairness guarantees.* We call this technique *batch unspooling.* Unspooling allows us to output transactions within a cycle *without waiting for the full cycle* to be seen. Nevertheless, we can guarantee that transactions within the same cycle will be output *contiguously*, i.e., no transaction from a later cycle will be ordered before all transactions from the current cycle are ordered. This allows us to **achieve the same fairness property** as Aequitas (in the scenario where final execution requires a total ordering) while providing standard liveness.

We note that figuring out when the current cycle ends brings back the same weak-liveness problem, but our technique of batch unspooling makes it so that this is no longer required.

4.3 `Themis` Description

We now describe our protocol `Themis` in detail. `Themis` can be bootstrapped from *any existing leader-based protocol* and will endow it with fair ordering with minimal design changes.

Overall design. Recall that standard leader-based protocols allow the leader to *unilaterally* choose its block proposal—only the validity of transactions and not their ordering is checked by the replicas. `Themis` will achieve fairness by providing a mechanism for replicas to also check the *ordering* in the proposal. To enable the leader to construct a fair ordering, all replicas will first submit their *local transaction orderings* to the leader. By local ordering, we mean transactions received at that replica ordered by their receive times.

Looking ahead, to enable batch unspooling across multiple leaders, we introduce our technique of *deffered ordering.* In line with this, we describe two algorithms to be executed by an honest leader: `FairPropose` for constructing (partial) proposals, and `FairUpdate` for any updates to previous proposals. In our core `Themis` design, the leader will send all the local orderings to all replicas to enable checking the fairness of its proposal. As a more theoretical alternative, we also later describe SNARK-Themis which leverages SNARKs to reduce the asymptotic complexity of this step.

Finally, we also describe an algorithm `FairFinalize`, that allows the replicas to extract a fair transaction ordering from fully specified proposals (we will elaborate on this later).

Local replica orderings. Before the leader constructs a proposal, each replica sends its local transaction ordering to the leader. Note that since the network is partially synchronous, the leader will need to work with only $n - f$ orderings. To enable the leader to construct a proposal, an honest replica i sends the following: (1) List_i containing the transactions received by i that are not part of any previous proposal, in the order that they were received (by i); (2) $\sigma_i(\mathsf{List}_i)$ which is i's signature on List_i. Further, to allow the leader to update previous proposals, i sends the following: (1) Update_i containing transactions from previous proposals that are not fully specified in the order that they were received (by i); (2) $\sigma_i(\mathsf{Update}_i)$ which is i's signature on Update_i. Notably, our protocol only requires *orderings* from the replicas, and not individual timestamps for transactions. In other words, `Themis` does not need to rely on synchronized clocks.

4.3.1 Constructing the Leader Proposal

The leader starts by checking the validity of the local replica orderings by checking their signatures. Then, to construct its proposal, the leader executes `FairPropose`($\mathcal{L}$), where $\mathcal{L}$ is a set of $n-f$ replica orderings. Abstractly, the goal of this algorithm is to propose as many transactions as possible while making sure that the exclusion of any transaction does not violate fairness.

Replica ordering notation. For a set $\mathcal{L}$ of $n-f$ orderings, we use $\mathrm{tx} \in_k \mathcal{L}$ (resp. $\mathrm{tx} \notin_k \mathcal{L}$) to denote that tx is present in at least (resp. less than) k orderings in $\mathcal{L}$. We use $\mathrm{tx} \prec_{(\mathcal{L},k)} \mathrm{tx}'$ to denote that tx appears before tx′ (this includes the case where only tx is present) in at least k orderings in $\mathcal{L}$. We use $\mathsf{Weight}_{\mathcal{L}}(\mathrm{tx},\mathrm{tx}')$ to denote the maximum value k such that $\mathrm{tx} \prec_{(\mathcal{L},k)} \mathrm{tx}'$.

Deferred ordering intuition. Abstractly, the final total transaction ordering can be thought of as being partitioned into cycles. While enabling batch unspooling across multiple leaders, we still need to guarantee that no transaction from a *later cycle* is output before all transactions from the current cycle are output. Towards this, based on the local replica orderings, the leader can split transactions into three types: *solid* , *blank* , and *shaded*. We use *non-blank* to label a transaction that is either solid or shaded . Very roughly, this delineation enables the following:

1. Solid transactions are the ones that are present in many replica orderings and can be sequenced within the current proposal. This property can be guaranteed for tx if $\mathrm{tx} \in_{n-2f} \mathcal{L}$.

2. Blank transactions are present in too few replica orderings and can be *excluded* from the current proposal. In particular, we can guarantee that if tx is blank, then it *cannot be part of a previous cycle* than any transaction in the current proposal; i.e., excluding it will not result in any unfairness. This happens when $\mathrm{tx} \notin_{n(1-\gamma)+f+1} \mathcal{L}$.

3. Shaded transactions take a more indeterminate form. These are transactions tx such that $\mathrm{tx} \in_{n(1-\gamma)+f+1} \mathcal{L}$ but $\mathrm{tx} \notin_{n-2f} \mathcal{L}$. In many cases, we can rule out the possibility of tx occurring in a previous cycle than all transactions in the current proposal, in which case tx can safely be excluded. In some cases however, current information does not allow for this to be ruled out—here, we must include the shaded transaction within the current proposal.

   Doing so can result in a somewhat different challenge. If two or more such shaded transactions are included, then the ordering *amongst them* may not be clear yet. Our deferred ordering technique applies exactly here. It allows proposing a partial ordering for shaded transactions which will be completed at a later point when they are present in $n-2f$ replica orderings. Notably, this happens only based on the actual network delay for these transactions to be received, and is therefore independent of other transactions.

Leader proposal algorithm. We now detail the `FairPropose` algorithm. Since some transactions may be partially ordered, we will have `FairPropose` output a *dependency* graph $\mathcal{G}$ that contains ordering dependencies within transactions in the current proposal. Later, through `FairFinalize`, replicas will extract out the final fair ordering from this graph.

The dependency graph $\mathcal{G}$ is constructed as follows: First, a vertex is added for each non-blank transaction. An edge (tx,tx′) is added to the graph whenever $\mathrm{tx} \prec_{(\mathcal{L},\mathsf{thresh})} \mathrm{tx}'$ where $\mathsf{thresh} = n(1-\gamma)+f+1$. Intuitively, this threshold is chosen so that an edge (tx,tx′) signifies that tx′ *cannot be in an earlier cycle* than tx. Only one of (tx,tx′) or (tx′,tx) is added however, i.e., if both conditions are satisfied, the one with the larger value will be added. Specifically, let $k = \mathsf{Weight}_{\mathcal{L}}(\mathrm{tx},\mathrm{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\mathrm{tx}',\mathrm{tx})$. If both $k,k' \geq \mathsf{thresh}$, then add the edge (tx,tx′) if $k > k'$ and the edge (tx′,tx) if $k' > k$; If $k = k'$, then one of the two edges can be added deterministically. Note that no edges are added when both $k,k' < \mathsf{thresh}$. When this happens, an edge will be added by a later proposal.

Next, we need to figure out which shaded transactions need to be included in the proposal and which ones can be excluded (recall that all solid transactions will be proposed). In particular, the proposal will contain exactly those shaded transactions that have a path (in $\mathcal{G}$) to a solid transaction.

Algorithm `FairPropose`($\mathcal{L}$)

`// Propose a fair ordering for new transactions.`

On input a set $\mathcal{L}$ containing the local orderings of $n-f$ nodes:

1. **Build Dependency Graph**
   - Create an empty graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$.
   - For each non-blank tx, add a vertex tx to $\mathcal{V}$.
   - For each $\text{tx}, \text{tx}' \in \mathcal{V}$, let $k = \mathsf{Weight}_{\mathcal{L}}(\text{tx}, \text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\text{tx}', \text{tx})$. If $k \geq n(1-\gamma)+f+1$ and $k \geq k'$ and $(\text{tx}',\text{tx}) \notin \mathcal{E}$, then add the edge (tx,tx') to $\mathcal{E}$.
   - Compute the condensation $\mathcal{G}^*$ and its topological sorting S.
2. **Output Fair Ordering**
   - Let V be the last vertex of S that contains a solid transaction.
   - Remove those tx from $\mathcal{G}$, that are in vertices after V in S.
   - Output $\mathcal{G}$.

Figure 4-1: Leader Proposal Algorithm

For this, we compute the condensation of $\mathcal{G}$ and topologically sort it. Let V be the last vertex in this sorting that contains a solid transaction. Then `FairPropose` outputs the resulting graph by removing all tx from $\mathcal{G}$ where $\mathsf{SCC}(\text{tx})$ occurs after V in the topological sorting; everything after this needs to be excluded. Details are given in Fig 4-1.

Proposal properties. The output of `FairPropose`($\mathcal{L}$) is guaranteed to contain all solid transactions in $\mathcal{L}$ and no blank transactions. Further, it contains exactly those shaded transactions that contain an outgoing path into a solid transaction within the dependency graph $\mathcal{G}$. Further, there is exactly one edge between any two vertices, except when both vertices are shaded (in which case there might be none). We prove this graph structure in Lemma 31. Future leaders cannot modify existing edges but will be able to add missing edges between shaded transactions. As mentioned before, our deferred ordering technique of proposing a "partial graph" is crucial as it allows us to avoid the weak-liveness problem.

To show that the graph proposal is consistent with batch-order-fairness , we need to show that any transaction exclusion does not violate fairness. In particular, we prove (in Lemma 33) that any excluded transaction that was excluded *cannot be in a previous cycle* than any transaction that was included. This implies that cycles are output *contiguously*, i.e., all transactions from the current cycle will be output before any transaction from a subsequent cycles.

Update algorithm. The algorithm `FairUpdate` allows a leader to add missing edges to a previous leader's proposal graph. For this, each replica i also sends the ordering Update_i that orders shaded transactions from previous proposals as seen by i. The update algorithm is quite simple: upon receiving $\mathcal{L}_{\text{updates}}$ containing $n-f$ such replica orderings, for all transactions tx and tx$'$ that were part of the same previous leader's proposal and do not currently have an edge between them, the edge (tx,tx') is added if $\text{tx} \prec_{(\mathcal{L},n(1-\gamma)+f+1)} \text{tx}'$. Once again, only one of (tx,tx') or (tx',tx) is added however exactly as in the `FairPropose` algorithm. The new edges are specified in an update list $\mathcal{E}_{\text{updates}}$. Intuitively, this allows all nodes to use these update edges to fill in the missing details within earlier proposals in order to compute the final fair ordering. We provide details in Fig 4-2.

The complete leader proposal can now be defined as the block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$. Replicas will use π to verify fairness; in SNARK-Themis , this can be done with a SNARK.

Algorithm `FairUpdate`($\mathcal{L}_{\text{updates}}$)

`// Update the ordering for previous proposals`

On receiving a set $\mathcal{L}_{\text{updates}}$ containing the local transaction orderings of $n-f$ nodes for previously proposed shaded transactions:

1. **Output Dependencies**
   - Let $\mathcal{E}_{\text{updates}} \leftarrow \emptyset$.
   - For all tx and tx$'$ that are part of the same leader proposal and between whom no edge has been proposed yet, let $k = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx},\text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx}',\text{tx})$. If $\text{tx} \in_{n-2f} \mathcal{L}_{\text{updates}}$, $k \geq k'$ and $k \geq n(1-\gamma)+f+1$ then add the edge (tx,tx$'$) to $\mathcal{E}_{\text{updates}}$.
   - Output $\mathcal{E}_{\text{updates}}$.

Figure 4-2: Update Algorithm

Consensus integration. The consensus design changes required are minimal. To enable the leader to construct its proposal, all replicas will first submit their local transaction orderings. The leader will now construct its proposal B and use the underlying consensus protocol to confirm it. Within this, replicas will also verify the *fairness* of the proposal (using π) before providing their signature. Since π contains all the replica orderings, the communication cost of the protocol will be $\mathcal{O}(n^2)$; this can theoretically be reduced to $\mathcal{O}(n)$ (when the leader is honest) by making π a SNARK instead. As a standard optimization, only the proposal hash needs to be signed. Note that other parts of the underlying protocol, e.g., leader election and view-change will remain exactly the same. Fig 4-8 (Appendix 4.8) provides a complete description.

Replica verification. Replicas can easily verify the fairness of the block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi)$. In particular, given $\pi = (\mathcal{L}, \mathcal{L}_{\text{updates}})$, replicas can simply run `FairPropose` and `FairUpdate` themselves to ensure that the proposed $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ are both correct.

4.3.2 Finalizing the Fair Ordering

Fully specified proposals. Recall that leaders can propose partial graphs for which missing edges will be added by future leaders. A proposal is *fully specified* if its graph is a *tournament*, i.e., there is exactly one edge between each pair of vertices. An important point here is that the confirmation of transactions will depend only on the network delay and does not suffer from the weak-liveness problem present in Aequitas. This is because missing edges in $\mathcal{G}$ will be added (making it fully specified) as soon as the transactions are received by the network and an honest leader is elected.

Transactions in fully specified proposals can be totally ordered and output by `Themis` . The algorithm `FairFinalize` describes how honest replicas can extract this final ordering.

Finalization algorithm. Given a sequence of proposals $B_1,...,B_l$ agreed upon by the underlying consensus protocol, the first step is to add any missing edges; in particular, if tx and tx$'$ are transactions in $B_i.\mathcal{G}$ such that $(\text{tx},\text{tx}') \in B_j.\mathcal{E}_{\text{updates}}$ for some $j > i$, then add the edge (tx,tx$'$) to $B_i.\mathcal{G}$. After the updates, let k be the largest index such that all $B_i.\mathcal{G}$ $(i \leq k)$ are now fully specified. Intuitively, transactions in these blocks will be ordered by `FairFinalize`.

To order transactions in $B.\mathcal{G}$, we first compute its condensation $B.\mathcal{G}^*$ by collapsing its SCCs and then find the topological sorting $[V_1, V_2, ..., V_c]$. Note that each of V_1 to V_c represent an SCC within $B.\mathcal{G}$. To output the final total ordering, we need to now order transactions within each V_i. As an optimization, transactions whose SCC as well as all SCCs that precede it contains only solid transactions can also be finalized immediately even before $B.\mathcal{G}$ is fully specified.

Observe that it is sufficient to order transactions in each V_i arbitrarily (since intuitively they represent cyclic dependencies) using an agreed-upon function; a similar technique is used in Aequitas. As

Algorithm `FairFinalize()`

Given a sequence of proposals $[B_1,B_2,...,B_l]$:

1. **Update Graphs**
   (a) For all B_i and transactions tx, tx$'$ in B_i that do not have an edge between them, if (tx,tx$'$) is in some $B_j.\mathcal{E}_{\text{updates}}$, then add that edge to $B_i.\mathcal{G}$.
   (b) Let k be the last index such that the graph $B_k.\mathcal{G}$ is a tournament.
   (c) Compute the condensation graphs of $B_1.\mathcal{G},\dots,B_k.\mathcal{G}$ and their topological sortings $S_1,...,S_k$.
2. **Retrieve Fair Ordering**
   (a) For each $S_i=[v_{i1},...,v_{il_i}]$ where $i\in\{1,...,k\}$, let H_{ij} be a Hamiltonian cycle of v_{ij} (pick one deterministically if there are multiple, e.g., by removing the edge with the smallest weight). Note that this exists since each v_{ij} is a strongly connected tournament.
   (b) Let tx_{il_i} be a solid transaction in each v_{il_i} (pick one deterministically if there are multiple), and let H'_{il_i} be the cyclic rotation of H_{il_i} so that the last transaction is tx_{il_i}.
   (c) The final transaction ordering for the block B_i is now given by $H_{i1},...,H_{i(l_i-1)},H'_{il_i}$.

Figure 4-3: FairFinalize Algorithm

a better alternative however, we show how to take advantage of useful hidden properties within the graph to provide stronger fairness guarantees, which we describe next.

Stronger fairness guarantees within SCCs. Instead of arbitrarily ordering transactions in an SCC, we will employ Hamiltonian cycles (a graph cycle that visits every vertex exactly once) to order transactions in a way that connects to the ordering in other SCCs.

An old result by Camion [47] shows that all strongly connected tournaments contain a Hamiltonian cycle. Note that although the problem of finding Hamiltonian cycles is NP-complete for general graphs, for tournaments specifically, they can be found in time linear in the number of edges [121].

Concretely, for each SCC C in $B.\mathcal{G}$, we first find a Hamiltonian cycle H_C containing all transactions in C. If multiple cycles exist, one of them can be chosen deterministically to be used for the total ordering. Now, the ordering for transactions within V_1 to V_{c-1} (of the topological sorting) is simply the sequence of the Hamiltonian cycles H_{V_1} to $H_{V_{c-1}}$. For the final SCC V_c, we will cyclically rotate the corresponding Hamiltonian cycle. Recall that our `FairPropose` algorithm guarantees that this contains at least one solid transaction, say tx. Now, to obtain the ordering for V_c, we rotate H_{V_c} so that tx is now the final transaction among those output.

The upshot is that we can now *cleanly connect* the boundaries of subsequent Condorcet cycles. Specifically, if $[\text{tx}_1,...,\text{tx}_l]$ is the final output ordering, then for any j, tx_j is received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes, regardless of whether tx_j and tx_{j+1} are in the same cycle. Notably, this is not achieved if transactions in a cycle are ordered e.g., alphabetically (as in Aequitas [105]).

A nice practical consequence of this property is strong resilience against a particular kind of front-running attack—specifically the one where the adversary wants to place its transaction *immediately before* an honest user's transaction.

4.3.3 Formal `Themis` Results

We will now formally define the properties satisfied by `Themis` in a partially-synchronous network when $n>\frac{4f}{2\gamma-1}$ where n is the number of nodes, f is the maximum number of adversarial nodes, and $\frac{1}{2}<\gamma\leq 1$ is the order-fairness parameter. The proofs are deferred to Appendix 4.8.1.

Theorem 13 (`Themis` fairness)**.** *At any time, let* $[\text{tx}_1,...,\text{tx}_l]$ *be the total transaction ordering output*

by `Themis` *. Then,*

1. *(*γ-batch-order-fairness *). The ordering can be linearly partitioned in batches satisfying* batch-order-fairness *. Concretely, there are indices* $1=i_1<\cdots<i_k=l+1$ *such that* $[C_1,...,C_{k-1}]$*, where each* $C_j=\{\text{tx}_{i_j},...,\text{tx}_{i_{j+1}-1}\}$*, satisfies* γ-batch-order-fairness *.*

   Importantly, (similar to Aequitas), the batches are minimal—they are of size one when there are no cyclic ordering dependencies. As a corrolary, receive-order-fairness is satisfied when there are no cycles.

2. *(Consequent-transaction fairness). For each* $i\in\{1,...,l-1\}$*, at least* $n(1-\gamma)+1$ *honest nodes received* tx_i *before* tx_{i+1}*.*

Theorem 14. `Themis` *guarantees SMR consistency and (standard) liveness.*

Other properties. In Appendix 4.8.2, we highlight that `Themis` is quite friendly to ex post facto auditing. We also remark that `Themis` requires only $n>3f+1$ nodes when dealing only with crash faults.

SNARK-based variant. Appendix 4.8.3 details a more theoretical design, `Themis` , which uses generic arguments of knowledge (specifically, SNARKs [34, 95] for NP) in a black-box way to verify computation. `Themis` achieves optimistic $\mathcal{O}(n)$ communication. Notably, we find that existing fair-ordering protocol designs cannot easily achieve the same even using SNARKS. This makes `Themis` the *first such fair-ordering protocol*; the communication here is asymptotically optimal and equivalent to state-of-the-art consensus protocols without fair-ordering guarantees.

4.4 Implementation and Benchmarks

We ran an extensive set of experiments for `Themis` that we detail in this section. In addition to the standard performance benchmarks for throughput and latency (Section 4.4.1), we also design a suite of fairness experiments (Section 4.5) which are useful in quantifying the extent of fair transaction ordering.

Implementation details. Our implementation for `Themis` is bootstrapped from the Hotstuff protocol [163]: a state-of-the-art leader-based protocol for the partially synchronous setting. For this, we started from the authors' open-source libhotstuff codebase [1] (which implements the $\mathcal{O}(n^2)$ version of the Hotstuff protocol). Our primary code changes were having the leader generate fair transaction sequences and having the replicas validate them. We use this implementation to benchmark the performance of `Themis` .

Remark 10 (Client communication)**.** *Traditional consensus research often ignores client communication: Transactions are typically abstracted to originate from the consensus nodes themselves. For fair-ordering protocols, however, it is important to model client transaction submission. In* `Themis` *(as in other fair-ordering protocols), an honest client submits its transaction to all nodes.*

Such models have been considered before. For instance, censorship resistance already requires submission to $f+1$ *nodes (since* f *corrupt nodes could just ignore the transaction). Furthermore, consistent broadcast protocols (e.g., [30]) also require transaction submission to all nodes (to obtain* $n-f$ *signatures).*

In the context of current blockchain infrastructure, clients often communicate through RPC nodes, which can create a centralization vector. Orthogonal to our context, RPC centralization is a well-known issue and efforts to decentralize the infrastructure have independently been proposed and deployed [17, 15]. We also note that for our purpose, a basic modification to client-side software (e.g., in Metamask) is sufficient to allow a client to send its transactions via multiple RPCs or directly to (permissioned) committee members.

4.4.1 Performance and Benchmarks

Experimental setup. For our performance experiments, we consider two settings: (1) Same Region: All nodes are in the same AWS region (us-east-2); (2) Geo-distributed: Nodes are distributed across 5 regions (us-west-1, us-east-1, ap-northeast-1, ap-northeast-2, eu-central-1) with an equal number of nodes in each region. To demonstrate scalability, we vary the total number of nodes n in the system from 5 to 100. Each node is run on an AWS EC2 C5.4xlarge instance with 16vCPUs and 32GiB memory. In addition, we utilize separate "client" nodes to generate and transmit transactions.

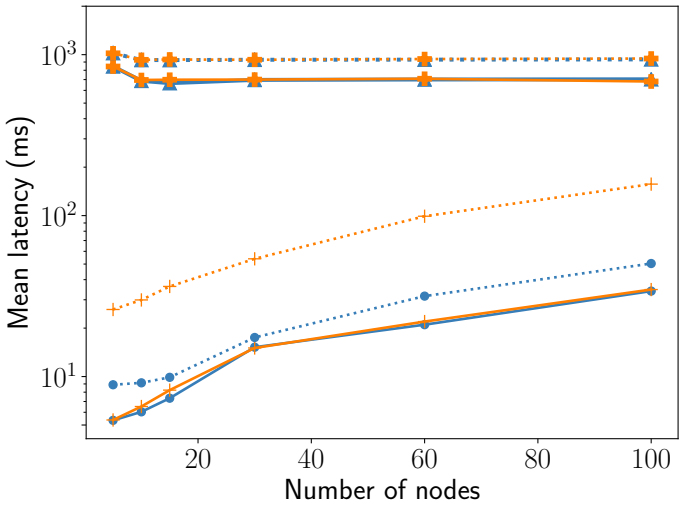


Figure 4-4: VDF.Eval(`input`,ek,τ) requires the number of iterations that VDF.Iterate should run.

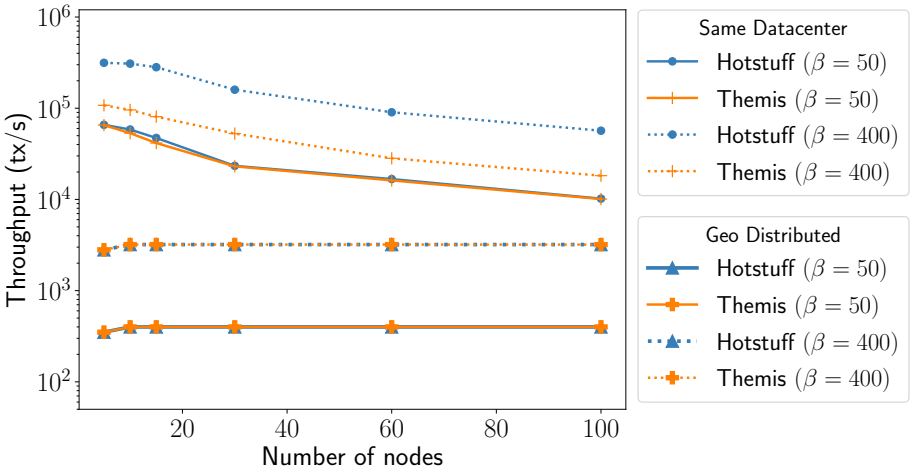


Figure 4-5: On the other hand, RandVDF.Eval(`input`,ek,`s`,`slot`) requires the expected number of number of iterations RandVDF.Iterate (denoted by `s`) must run.

Figure 4-6: Performance comparison of `Themis` and Hotstuff in both the same-datacenter and geo-distributed settings.

Latency and throughput. We report on the latency and throughput of `Themis` and compare them to Hotstuff run as a baseline (see fig 4-6). We experimented with two blocksize (denoting the number of transactions in a proposal) parameters $\beta = 50$ and $\beta = 400$.

Overall, `Themis` provides very comparable performance to Hotstuff. Notably, we found that both

latency and throughput of Themis scale in the same way as Hotstuff as the number of nodes increase. Using a larger blocksize increases the performance gap between the two protocols; this is because Themis needs to build a graph with $\mathcal{O}(\beta^2)$ edges. In the single datacenter setting, this gap was found for blocksize $\beta = 400$. This difference is not fundamental however, and we highlight that the computation can be parallelized and therefore, by using more cores per node, the performance of Themis can be made comparable to Hotstuff even for a large β. An optimized implementation of the graph algorithms used by Themis could also further boost the performance.

More importantly, we found that this difference vanishes in the geo-distributed setting due to already larger communication latency. In fact, we found that performance of both systems was identical even for very large blocksizes ($\beta = 1200$).

In essence, we expect the performance of Themis to be sufficient for most applications that require its fairness properties.

Performance comparison to other fair-ordering protocols. We do not directly compare the performance of Themis to other fair-ordering protocols; this is primarily due to lack of comparable implementations. For instance, Aequitas and the protocol from [44] have not been implemented, while an open-source implementation of Wendy only provides simulations and is not yet integrated with the consensus layer. Pompē is implemented using Hotstuff as the underlying protocol; their benchmarks also show comparable performance to Hotstuff (similar to Themis), and even better performance in some geo-distributed deployments, due to a fast *ordering* phase that precedes the main consensus layer. However, since this ordering phase itself creates censorship issues within Pompē , a performance comparison would not be on an equal footing.

4.5 Suite of Fair Ordering Experiments

To analyze and compare different fair-ordering definitions and protocols on the same footing, we propose a general suite of *fairness experiments*. Through these, we compare the fairness definitions from [168, 110] as well as receive-order-fairness , and batch-order-fairness . We also compare Themis to other fair-ordering protocols.

Comparison axes. Our analysis comprises primarily of three axes—each targeted at specific setting or attack vector. We start by considering an *ideal* setting where all nodes are honest to understand the best-case scenario (Section 4.5.3). Our next two experiments simulate adversarial environments. In Section 4.5.4, we analyze susceptibility to *insertion* (e.g., front-running) attacks. In Section 4.5.5, we evaluate robustness to *reordering*—to what extent adversarial nodes can influence the final ordering by reordering honest user transactions.

As a related analysis point, in Section 4.5.6, we also evaluate censorship-resistance from a formal standpoint; here, we show a subtle censorship issue with Pompē [168].

4.5.1 Fair-Ordering Definitions

Fair ordering definitions from [168, 110]. Zhang et al. [168] and Kursawe [110] consider similar definitions of fairness which we consolidate into a single property—*fair separability*.

Definition 20. *If all honest nodes receive* tx *before any honest node receives* tx′*, then all honest nodes output* tx *before* tx′*.*

This property is called *timed-relative-fairness* in [110]. It is also stated as a desideratum in [87] although no protocols are given. The *ordering-linearizability* notion considered in [168] is similar except that it is only applied when tx and tx′ are both output by the protocol. In particular, ordering-linearizability considers it acceptable if only tx′ is ouput and tx is not, even when the antecedent of the above definition is true, i.e., all honest nodes received tx before any honest node received tx′. By doing so, the Pompē protocol is able to ensure that the delivery of tx′ is not held back by tx even when tx is stuck in a slow network. However, by making this tradeoff, another problem is introduced: a network

adversary or even a Byzantine leader node is now able to *censor* a specific transaction from being delivered; this is potentially far more problematic, especially for our primary motivation of DeFi. We discuss this further in Section 4.5.6.

Comparing batch-order-fairness **on an equal footing.** While batch-order-fairness allows transactions to be output together, this is only done within the same cycle, and even within a batch, a total ordering will be enforced later for execution. Therefore, to quantify the strength of batch-order-fairness in the most conservative way, in all our comparisons, we consider the final *total execution ordering* it guarantees, i.e., exactly the fairness property of `Themis` .

For instance, if $\mathcal{A}$ can adversarially get tx *executed* before tx$'$ (e.g., through frontrunning or order-manipulation), we count it as a victory for $\mathcal{A}$ and a failure of batch-order-fairness and our protocol. In fact, even if $\mathcal{A}$ is able to place tx and tx$'$ into the same cycle when they should not have been, we will still consider it a success for the adversary. We emphasize that even with this conservative approach, our results highlight the strength of batch-order-fairness and `Themis` in preventing order-manipulation attacks.

4.5.2 Simulation Environment

We created an environment to simulate the creation and network broadcast of transactions to better understand the effect of different parameters. This is useful for both our ideal setting (Section 4.5.3) and adversarial reordering (Section 4.5.5) experiments. Transactions are generated by a *sending process* with the time delays between consecutive transactions sampled from the distribution $\mathsf{GenerationDist}$. For each transaction, we simulate when it would reach different consensus nodes by sampling the network latency from a distribution $\mathsf{NetworkDist}$. Let $\mathsf{Send}(\text{tx})$ denote the time that tx was generated and $\mathsf{Recv}(i,\text{tx})$ denote the time that tx is received by replica i.

We instantiate each distribution as an exponential distribution, which is standard in networking literature. The rate of generation or arrival of messages is modeled as a Poisson process, which is equivalent to the intervals between messages (i.e., $\mathsf{GenerationDist}$) being an exponential distribution. Separately, the network delay is also usually modeled as an exponential distribution. We set $\mathsf{GenerationDist}$ to the exponential distribution $\mathsf{Exp}(1/\mu)$ with mean $\mu=1$, and chose $\mathsf{NetworkDist}$ to be *independently distributed* with mean $r\mu=r$ for a *network ratio* parameter r.[3]

r represents how quickly new transactions are created compared to their propagation time. It serves as a proxy for how far apart nodes are from one another; a small r ($\ll 1$) captures a setting where all nodes are in the same local network, while a larger r (say 10 or 100) is typically more reflective of a geo-distributed setting.

We highlight that even when the network delay is very large, transaction latencies with smaller r can be approximated by, abstractly, setting a coarser *granularity* for fairness. We include a discussion in Appendix 4.9. In our experiments, we assume the finest possible granularity since this is the most challenging setting, but we note that in some cases, a coarser granularity may be acceptable.

Transaction comparisons. It is most fruitful to analyze the ordering for transactions sent around the same time. The scenario where tx$'$ is sent after tx has already been ordered, for instance, is not particularly interesting. Therefore we take the approach of studying sets of temporally clustered transactions, rather than workloads of extended duration; we will use sets of 1000 such transactions.

4.5.3 Ideal Setting Comparison

We first seek to understand the utility of different fair-ordering properties *in an ideal, non-adversarial setting*. Here, the only influence is from how far nodes are and any randomness in network propagation. In this ideal setting, we ask how close we can get to the magical first-in-first-out property of ordering transactions based on when they were *sent*. We only summarize the results below, and defer the complete details to Appendix 4.9.

[3]The choice of $\mu=1$ is w.l.g. since exponential distributions satisfy the scaling property — If $X\sim\mathsf{Exp}(1/\mu)$, then $kX\sim\mathsf{Exp}(1/k\mu)$.

Ideal ordering summary. We vary r from 10^{-2} to 10^3, and measure the number of transaction pairs that are correctly accounted for by different fairness definitions, i.e., their ordering is consistent with the sending process: tx is ordered before tx′ when $\mathsf{Send}(\text{tx}) < \mathsf{Send}(\text{tx}')$. To quantify the usefulness of batch-order-fairness conservatively, we consider it a failure of the definition if transactions end up in the same cycle. Overall, we still find that both receive-order-fairness and batch-order-fairness ensure closer to ideal ordering than fair separability for all values of r.

Of course, as r grows (i.e., the network delay gets larger), the order of transactions received at the nodes will be vastly different than the send ordering, i.e., the number of transaction pairs ordered correctly will drop to zero. Still, the interest in this experiment is understanding how quickly this drop takes place for different definitions. Further, observe that having a larger n makes each definition fare better.

We also find that receive-order-fairness and batch-order-fairness perform identically except for when $\gamma \approx 0.5$. The deviation between the two definitions occurs due to (strong)-Condorcet cycles (since we are in an ideal setting); intuitively, this shows that these cycles are infrequent. To underscore this, we further investigate these cycles; we find that cycles are rare, and are of small length even when they do arise (see Fig **??** in Appendix 4.9). This effectively demonstrates that even though the stronger receive-order-fairness is impossible without network synchrony, in many practical settings, the performance of batch-order-fairness is almost identical.

4.5.4 Network Level Insertion Attacks and Frontrunning

We now analyze how robust a fairness definition is to network-level *frontrunning*, which is arguably the core ordering issue in today's networks. We define network-level frontrunning as a node Y being able to perform the following attack: On receiving tx from another node X, it attempts to create a new transaction tx′ and get it sent to other nodes in an attempt to get tx′ ordered before tx. Note that this frontrunning can take place before transactions are received by all nodes, i.e., even before the consensus protocol begins.

While causal ordering or privacy techniques can help hide transaction data before ordering, thus thwarting targeted frontrunning (i.e., based on transaction content), they do not help against problems such as metadata (e.g., IP address) leakage, or non-targeted frontrunning (i.e., based only on the transaction existence rather than its content). Therefore, quantifying network frontrunning remains important. Notably, our experiment measures frontrunning protection even in cases when transaction privacy is not sufficient. Nevertheless, we note that all fair ordering protocols can incorporate privacy as a complementary protection—e.g., as a backstop against fair-ordering failures in extreme settings such as full adversarial network control. We also emphasize that our experiment can be easily adapted to other network-level attacks like backrunning, sandwiching, and priority attacks (e.g., first in line for an ICO [122]).

Formal analysis in a natural network setting. As a concrete result, we show how `Themis` as well as the order-fairness definitions more broadly prevent frontrunning in a natural network setting where triangle inequality is respected. Abstractly, we prove that in this model, an adversary cannot force a frontrunning transaction tx_{adv} into the same Condorcet cycle as an honest transaction tx, and consequently will not be able to get tx_{adv} executed earlier. This serves to complement our experimental analysis of frontrunning in real network settings. The full details are given in Appendix 4.9.2.

Experiment details. We used a real network rather than a simulation for this experiment to understand the possibility of frontrunning in practice. Our experiment studies frontrunning at the network layer; we do this to separate out the impact of a network adversary from adversarial transaction reordering through controlling consensus nodes (studied in Section 4.5.5). Through measurement of communication latency between nodes, our goal now is to find the number of *frontrunnable pairs*—i.e., (A,B) such that B is able to frontrun transactions originating from A. While we experimentally find the number of such pairs, we emphasize that in practice, the success of frontrunning also depends on other factors—for instance, the expected profit of a particular frontrunning strategy, the variance in network latency, the presence of competing entities etc.

Frontrunnable pairs. We now describe frontrunnable pairs for the fairness notions considered. For fair separability, frontrunning is possible if there exists nodes A,B,C,D (where A, B, and C are distinct, D is different from A and B but potentially the same as C) such that $\mathsf{ping}(A,B)+\mathsf{ping}(B,C)<\mathsf{ping}(A,D)$. When this happens, (A,B) is frontrunnable since B can receive tx from A (working as a client) and forward its own adversarial transaction tx' to C before tx was received by all honest nodes. Here, fair separability will not apply since the antecedent is violated—there is an honest node (C) that received tx' before some other honest node (D) received tx.

For (receive/batch)-order-fairness, (A,B) is frontrunnable if there are at least $n(1-\gamma)+f+1$ nodes C such that $\mathsf{ping}(A,B)+\mathsf{ping}(B,C)<\mathsf{ping}(A,C)$. This is because tx sent by A is no longer receiver before tx' by at least $\gamma n-f$ nodes. Frontrunning success is guaranteed if there are more than $\gamma n-f$ such nodes C. Observe that each satisfying (A,B,C) corresponds to breaking triangle inequality. Therefore, the possibility of frontrunning here for (A,B) is synonymous to triangle inequality being violated in the network for a *non-trivial* (at least $f+1$) number of nodes. We find that is quite unlikely to hold for real-world networks. For order-fairness, we report on both the $\gamma=1$ case (for which triangle inequality needs to be violated $f+1$ times for the pair to be frontrunnable), and the optimal case (for which it needs to be violated $>n/2$ times).

4.5.5 Robustness to Adversarial Reordering

In this section, we evaluate to what extent adversarial nodes can influence the final transaction ordering through *reordering*, i.e., by claiming within the protocol to have received user transactions in a different order than they were actually received.

Intuition. All fair-ordering properties guarantee some robustness to adversarial reordering. For instance, when all honest nodes receive tx before any honest node receives tx', fair separability guarantees that tx will be output before tx' no matter what the adversary does. Yet this says nothing about *how often* the fairness property will apply for transaction pairs or what happens when the property *does not apply* (e.g. is the fairness all or nothing or does it smoothly degrade?). Understanding this will paint a more complete picture of the overall robustness of a protocol to adversarial reordering.

Dependence on transaction *closeness*. Zhang et al. [168] show that it is impossible to make the final ordering completely independent of the adversary (since intuitively, adversarial nodes are indistinguishable from honest nodes). Still, ideally, a good ordering protocol restricts adversarial influence only to transactions that are already "very close" in the honest ordering — close in the sense that even their honest ordering could have been inverted by small network delays. Effectively, this would signify that the impact of the adversary is similar to that of small network fluctuations.

As a proxy for "closeness," we use the distance metric $\mathsf{Dist}(\mathrm{tx},\mathrm{tx}')=|\#_{(\mathrm{tx}<\mathrm{tx}')}-\#_{(\mathrm{tx}'<\mathrm{tx})}|$ where $\#_{(x<y)}$ denotes the number of nodes that received x before y. Dist varies from (n mod 2) to n in increments of 2. A small Dist implies a "fragile" pair, where the number of nodes that received tx earlier is roughly the same as the number that received tx' earlier, while a large Dist means that one of the transactions was received earlier by a large number of nodes.

Adversarial strategy. Since the space of possible adversarial strategies is practically unbounded, we consider just one specific yet powerful attack in which the adversary will attempt to flip the ordering of transactions. This adversarial strategy is quite simple: an adversarial node simply flips the ordering of all transactions it received as input. For instance, if it received transactions in the order $[\mathrm{tx}_1,\ldots,\mathrm{tx}_l]$, it claims to have received them in the order $[\mathrm{tx}_l,\ldots,\mathrm{tx}_1]$. Despite the simplicity, this should be a reasonable strategy for attempting to reverse the final ordering for many transaction pairs. It generalizes attempting to flip the ordering of a single transaction pair. Still, we acknowledge that better adversarial strategies likely exist and fair ordering protocols should also be tested against them. We leave this for future work.

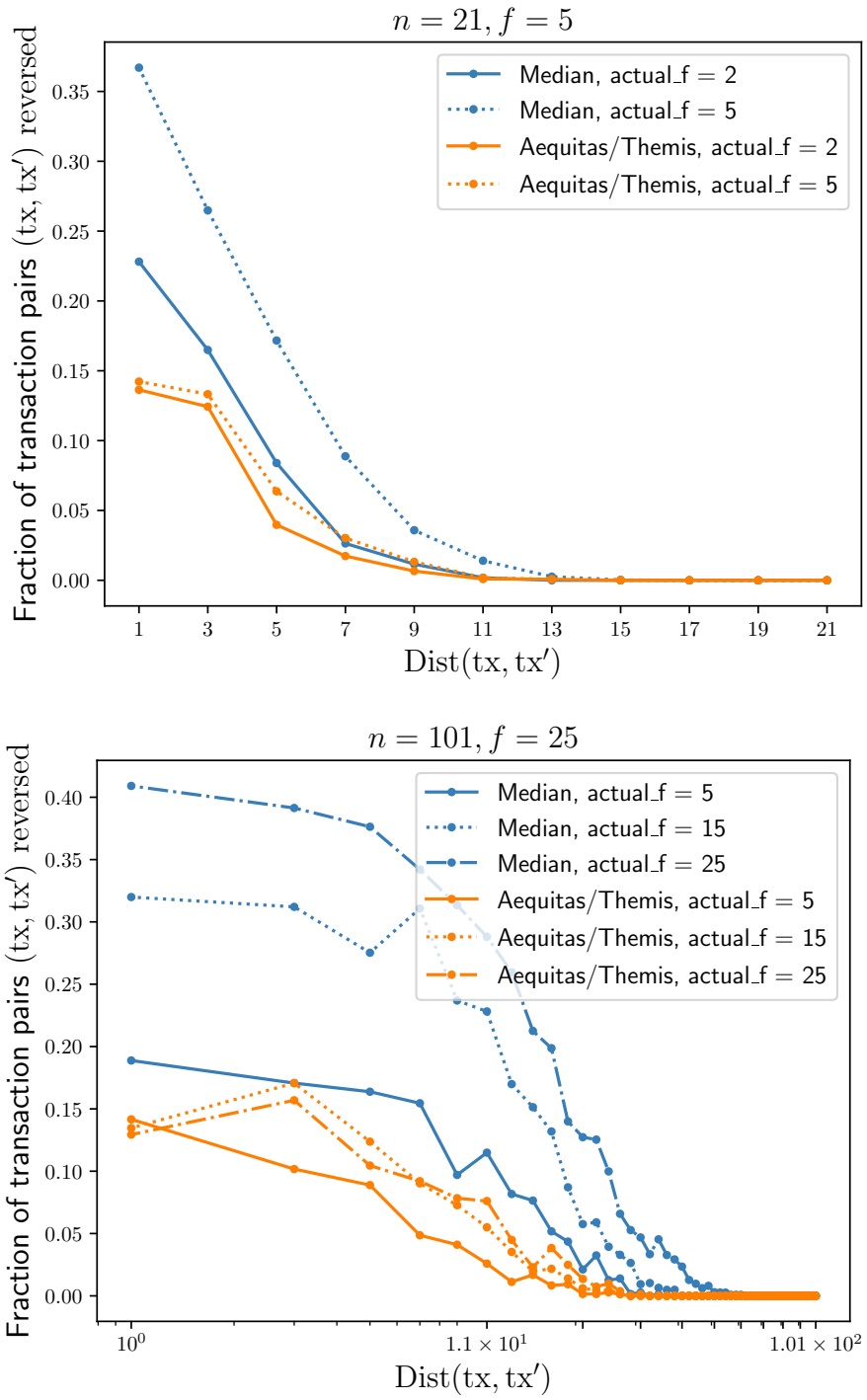


Figure 4-7: Adversarial influence on transaction ordering

Measurement. Given our adversarial strategy of transaction flipping, we now find how frequently it enables the adversary to reverse the ordering (w.r.t. the honest ordering) of a transaction pair with a given distance. As a more accurate comparison, for `Themis` , even if the adversary is able to put transactions into the same cycle when they should not have been, we consider as a failure for our protocol.

We consider two settings: $n = 21, f = 5$ and $n = 101, f = 25$, and vary the *actual* number of corruptions. In each instance, we find the fraction of transaction orderings the adversary successfully

inverted. Fig 4-7 shows the results. We compare Themis ($\gamma = 1$) to a simple *median* protocol which sorts transactions by their median timestamp. Note that this protocol abstracts exactly the relevant fairness component of Pompē [168] and Wendy [110].

Reordering insights. We first notice that for large $\mathsf{Dist}(\mathrm{tx}, \mathrm{tx}')$ (≥ 13 for $n = 21$, and ≥ 71 for $n = 101$), there was no successful reordering against either protocol, even with the maximum number of corruptions (for Themis , this was true for all $\mathsf{Dist} \geq 11$ for $n = 21$ and ≥ 29 for $n = 101$). Note that the X-axis in the $n = 101$ figure is in log-scale to better highlight reordering for transactions pairs with small distance (since reordering was absent for large Dist).

The fraction of reversed transaction pairs drops sharply as Dist gets larger, which shows both fair ordering protocols working as expected. For small Dist, the transactions are so close that even non-adversarial random network delays can reverse their ordering, and thus it is unreasonable to expect resilience to adversarial strategies.

Still, overall, we found that for a given number of actual corruptions, it is much easier to reverse the ordering for a median timestamp protocol than for Themis . In fact, in our experiment, it was easier to reverse the ordering against the median timestamp protocol even through a small number of corruptions than it was against Themis using the maximum number of corruptions. Further, we also note that for this specific adversarial strategy, the additional gain against Themis through extra corruptions is small.

4.5.6 Censorship Resistance and Attacks

Censorship resistance is defined as the property that if an honest client sends a transaction to the consensus nodes, then it will eventually be output by the protocol. While many existing protocols provide censorship resistance (e.g., [126]), we emphasize that on its own this does not guarantee that the transaction *ordering* is "fair."

We find that some fair-ordering designs can have inherent tradeoffs between liveness and censorship resistance; this is seen when an earlier phase is used to first *order* transactions prior to consensus.

Censorship attack on [168]. Pompē [168] uses a pre-protocol (an ordering phase prior to consensus) to compute the median timestamp for transactions, with the fair ordering being taken according to the ascending order of the median timestamp. For $n \geq 3f + 1$, as long as (tx,tx′) satisfies the antecedent of fair separability (Definition 20), the median of tx will be smaller than that of tx′. In other words, as long as both transactions *complete the ordering phase*, tx will be ordered earlier by the protocol; this condition is explicitly stated in the *ordering linearizability* definition used in Pompē . Unfortunately, this results in censorship resistance being guaranteed only for transactions that pass the pre-protocol phase.

For Pompē , in a partially synchronous network, an adversary can delay sending the median timestamp for tx long enough until tx′ is delivered. If this happens, since the rest of the output is fair, tx can never be output by the protocol. Concretely, Pompē assumes a known upper bound Δ_2 on the time for an honest node to complete the ordering phase; Δ_2 is the maximum time a transaction in the consensus phase waits for any potential transaction from earlier to finish the ordering phase. When this bound is not satisfied, while safety is not broken, earlier transactions are now essentially censored since the consensus phase has moved on to a later timestamp even though they were timestamped during the ordering phase. In order words, to achieve censorship resistance, Pompē needs to assume a known synchronous bound *throughout* the protocol on the time for an honest node to complete the ordering phase, which cannot be assumed in the partially synchronous setting. The synchrony requirement is also especially of concern in the common setting where clients (rather than nodes) submit transactions since the bound would now have to involve the client's connection.

While this was partially acknowledged in [168] as a possible way a Byzantine node could selectively chose which of its transactions to submit to the consensus phase, the stronger censorship attack vector was not identified. In the context of DeFi, both selective disclosure and censorship are far more severe. This also surfaces an inherent tradeoff between liveness and censorship resistance within such protocols. Since the client (or the protocol node in charge of the transaction) is in charge of accumulating timestamps from the nodes in the ordering phase before proceeding to the consensus phase, as [168] notes, it is not possible to determine whether the transaction was sent to the consensus phase in time or not. In

fact, as a result, a view change also cannot be triggered in order to restore censorship resistance. Thus either there will be a liveness failure (by potentially waiting forever), or a lack of censorship resistance.

From our analysis, the core reason for this is the separation between the pre-protocol that computes the median timestamp and the consensus phase, along with the design that a single node is responsible for moving a given transaction from the former to the latter phase. Here, due to the presence of Byzantine nodes, it is impossible to ensure that any transaction that has been timestamped at the protocol nodes also makes its way into the consensus phase.

In contrast, `Themis` , along with `Wendy` and Aequitas, are not susceptible to such censorship; if tx is received by all honest nodes, it will not be censored, and moreover will be sequenced fairly.

4.6 Related Work

While our experimental suite contains several useful comparisons, we use this section to highlight a few more nuances.

Causal ordering and privacy. Prior work in classical distributed systems [42, 144] had considered a limited notion of preventing reordering based on *transaction content*, but as mentioned earlier (and also identified by [105, 102, 110]), this is not robust against metadata leakage, collusion with protocol nodes, and order-manipulation attacks that do not rely on the content of honest transactions.

Informally, causal ordering uses threshold encryption to hide transaction data before its position is finalized in the total ordering. In a similar spirit, other recent protocols have proposed to use alternative cryptographic privacy mechanisms such as commit-and-reveal schemes [**?**], time-lock encryption, VDFs [**?**], or MPC among the protocol nodes [**?**]. These protocols have the same drawbacks as causal ordering when compared to the line of work on fair ordering protocols. In addition, these protocols are somewhat ad hoc, to our knowledge lacking formal analysis on their properties.

Often, transactions within an epoch are also randomly ordered. As noted in, e.g., [102], this is susceptible to flooding attacks (e.g., [122])—by spamming the network with many adversarial transactions, with high probability, at least one of them will be unfairly ordered ahead of the honest transaction and succeed in extracting profit from it.

We note however that confidentiality-based techniques can nicely complement fair ordering protocols; confidentiality can protect transactions against ordering attacks at the network layer, while fair ordering does so at the consensus layer.

Further comparison to Aequitas [105]. We find interesting insights in the structure of the dependency graph constructed by Aequitas and `Themis` . A brief discussion appears in Appendix **??**.

Comparison to Cachin et al. [44, 43]. Concurrent to this work, Cachin et al. [43] defined κ-differential-order-fairness (for $\kappa \geq 0$), which states that if $b(\mathrm{tx}_1,\mathrm{tx}_2) > b(\mathrm{tx}_2,\mathrm{tx}_1)+\kappa+2f$, where $b(x,y)$ denotes the number of honest nodes that receive x before y, then tx_1 will be delivered no later tx_2. In Theorem **??**, we prove that this definition is simply a reparameterization of batch-order-fairness (and therefore equivalent). The original protocol from [43] contained a major bug which was later fixed in a subsequent revision [44]. Unfortunately, this revised protocol offers significantly weaker security; the proof for liveness only works when *all nodes are honest*, which makes it less useful than `Themis` in most practical settings.

4.7 Conclusion

We introduced `Themis` , a consensus protocol that satisfies a strong fair transaction-ordering property, and has efficiency comparable even to state-of-the-art consensus protocols that lack fair-ordering properties. We also introduced a new systematically designed suite of experiments related to fairness to evaluate and compare `Themis` to the recent exciting line of work on fair-ordering protocols. These experiments show that the fairness property satisfied by `Themis` is stronger in practice than alternative notions.

Complete `Themis` Description

Round leader protocol:

1. Wait for $n-f$ correctly signed lists $(\mathsf{List}_i, \mathsf{Update}_i)$ from replicas. Let $\mathcal{L} = \{\mathsf{List}_i\}$ and $\mathcal{L}_{\text{updates}} = \{\mathsf{Update}_i\}$
2. Run $\texttt{FairPropose}(\mathcal{L})$ and $\texttt{FairUpdate}(\mathcal{L}_{\text{updates}})$ to get $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ respectively.
3. Run the underlying $\mathsf{Consensus}$ protocol to agree on a valid $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$ and append it to the list of agreed upon blocks.

Replica protocol (for N_i):

1. Let List_i denote the ordered list of transactions received by N_i that are not yet part of any leader proposal. Let Update_i denote the ordered list of transactions received by N_i that are correspond to vertices with missing edges in some previous leader proposal.
2. Send $\mathsf{List}_i, \mathsf{Update}_i$ to the current round leader and verify that the leader's response is a valid proposal (by checking the SNARK proofs).
3. Participate in the underlying $\mathsf{Consensus}$ protocol to agree on a block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}})$ and append it to the list of agreed upon blocks.

Constructing the final ordering (for N_i):
At any point, N_i can do the following to append new transactions to its final ordering:

1. Let B_j be the first block whose transactions have not yet been output in the final ordering by N_i, and let B_k be the final block.
2. Run `FairFinalize` with $B_j, ..., B_k$ to obtain an ordering $[\text{tx}_p, ..., \text{tx}_q]$ of transactions.
3. Append $\text{tx}_p, ..., \text{tx}_q$ to the final transaction ordering.

Figure 4-8: Complete Description of `Themis`

4.8 `Themis` Details

Fig 4-8 contains a complete description of `Themis` from the perspective of both the leader and the replica nodes.

4.8.1 Proofs for `Themis`

In this section, we formally prove the theorem statements from the main text. We begin with some helpful lemmas.

Lemma 31 (Dependency Graph Properties)**.** *The following hold for the output of the algorithm* $\texttt{FairPropose}(\mathcal{L})$*: (1) It contains all solid transactions; (2) For all solid* tx *and non-blank* tx' *contained in the output, exactly one of the edges* (tx,tx') *and* (tx',tx) *is present in the output graph.*

Proof. Let $\mathcal{G}$ be the dependency graph constructed by the algorithm $\texttt{FairPropose}(\mathcal{L})$. It is easy to see why (1) holds since the final graph output only removes those SCCs from $\mathcal{G}$ that do not contain a single solid transaction.

To prove (2), first notice that since tx is solid and tx' is non-blank , at least $n-2f > 2n(1-\gamma)+2f$ orderings in $\mathcal{L}$ contain at least one of tx or tx', i.e., order the two transactions. Therefore, at least one of $\text{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \text{tx}'$ or $\text{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \text{tx}'$ holds. Since only one edge is added even when both holds, we conclude that exactly one of the edges (tx,tx') and (tx',tx) is contained in the final graph output. □

Lemma 32. *If* `FairPropose(`$\mathcal{L}$`)` *outputs a solid transaction* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then there are at least* $n(1-\gamma)+1$ *honest nodes that have received* tx *before* tx′.

Proof. If tx′ is blank , since tx′ is present in at most $n(1-\gamma)+f$ orderings, we have $\mathrm{tx} \prec_k \mathrm{tx}'$ where $k = n-2f-n(1-\gamma)-f = \gamma n-3f > n(1-\gamma)+f$ since $n > \frac{4f}{2\gamma-1}$. Equivalently, since at most f of those are adversarial, it holds that more than $n(1-\gamma)+1$ honest nodes have received tx before tx′. □

Lemma 33 (Contiguous Cycles)**.** *If* `FairPropose(`$\mathcal{L}$`)` *outputs* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then either* tx *is an earlier cycle than* tx′ *or* tx *and* tx′ *are in the same cycle. Further, if* tx′ *is received before* tx *by* γn *nodes, then* tx *and* tx′ *are in the same cycle.*

Proof. We first show that if tx is output and tx′ is not, then it cannot be the case that tx′ is in an earlier cycle. To see why, assume for contradiction that tx′ is an earlier cycle. This means that there is no sequence of transactions $[\mathrm{tx}=\mathrm{tx}_0,\mathrm{tx}_1,...,\mathrm{tx}_l=\mathrm{tx}']$ such that for each j it holds that tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. But this contradicts the fact that tx was output and tx′ was not, and therefore we conclude that tx′ cannot be in an earlier cycle.

Now, for the case when tx′ is received before tx by γn nodes, note that tx cannot be solid within the leader proposal since this would imply that there are at least $n(1-\gamma)+1$ honest nodes that have received tx before tx′ (from Lemma 32), which contradicts the given condition. Therefore tx must be shaded and consequently, it was included in the proposal since there is a sequence of transactions $[\mathrm{tx} = \mathrm{tx}_0, \mathrm{tx}_1, \ldots, \mathrm{tx}_l]$ such that the graph contains each of the edges $(\mathrm{tx}_i, \mathrm{tx}_{i+1})$ and tx_l is a solid transaction. This also means that tx_l was received before tx′ by at least $n(1-\gamma)+1$ honest nodes.

Now, since tx′ is received before tx by γn nodes, this directly implies that $[\mathrm{tx}_0,\mathrm{tx}_1,...,\mathrm{tx}_l,\mathrm{tx}']$ is a cycle. □

We are now ready to prove the main theorems for `Themis` .

Proof of Theorem 13. To show batch-order-fairness , consider tx and tx′ such that tx was received before tx′ by at least γn nodes. Now, we follow a few cases:

(Case 1) At some time, the current correct leader proposal includes tx and tx′ is not included in this or any earlier proposal. Now, by Lemma 33, we know that tx′ must be in the same or later cycle as tx.

(Case 2) At some time, the current correct leader proposal includes tx′ and tx is not included in this or any earlier proposal. Since we are also given that tx was received before tx′ by γn nodes, by Lemma 33, we know that tx′ must be in the same cycle as tx.

(Case 3) At some time the current correct leader proposal includes both tx and tx′. Since tx was received before tx′ by γn nodes, we know that there is an edge from tx to tx′ within the dependency graph. Therefore, once again, either tx′ will be output in the same cycle as tx or a later one.

The above cases show that the final transaction ordering final transaction ordering can be split into contiguous cycles. In other words, either tx will be output before tx′ or some contiguous transaction sequence containing both tx and tx′ will be a cycle. Consequently, there are indices $1=i_1<\cdots<i_k=l+1$ such that $[C_1,...,C_{k-1}]$, with batches $C_j=\{\mathrm{tx}_{i_j},...,\mathrm{tx}_{i_{j+1}-1}\}$, satisfies γ-batch-order-fairness .

Now, to show consequent-transaction fairness, in the output ordering, consider transactions tx_j and tx_{j+1} where tx_{j+1} is output immediately after tx_j. The following cases arise:

(Case 1) Both transactions were part of the same initial leader proposal. Now, if they were part of the same SCC, then since the output order is a Hamiltonian cycle, $(\mathrm{tx}_j,\mathrm{tx}_{j+1})$ is an edge in the graph; in other words, tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. On the other hand, if tx_j was in an earlier SCC, then $(\mathrm{tx}_j,\mathrm{tx}_{j+1})$ is already an edge. Since tx_j was output earlier, it cannot be in a later SCC.

(Case 2) tx_{j+1} was proposed in the block after tx_j was proposed. This means that, in some valid leader proposal, tx_j was present but tx_{j+1} was not, and further tx_j was the last transaction in the previous proposal, which implies that tx_j was a solid transaction. Therefore, we can directly conclude the result by Lemma 32.

(Case 3) Note that it not possible for tx_j to be proposed in a block after tx_{j+1} since the final ordering contains tx_j first.

□

It is easy to see that, similar to Aequitas, the batches are minimal in the sense that they are of size one when there are no cycles. Consequently, in such a case, `Themis` satisfies the stronger receive-order-fairness notion. We now show that `Themis` satisfies consistency and (standard) liveness.

We now prove that `Themis` satisfies the standard SMR properties.

Proof of Theorem 14. Consistency is a direct consequence of the consistency of the underlying consensus algorithm. For liveness, consider a transaction tx that has been received by all nodes, i.e., it will be present in at least $n-2f$ local replica orderings sent to the leader. In other words, tx is solid , and will be included in the leader's proposal. Note that the final ordering of tx only depends on earlier (shaded) transactions that have been proposed by the current or an earlier proposal (since this can cause edges to be missing in the graph proposal). Note that the final ordering of tx no longer depends on any transactions that are not yet proposed. This means that as soon as the edges between the previously shaded transactions are added (this will happen when the shaded transactions are received by enough nodes which is only dependent on the network delay). Consequently, `Themis` achieves standard liveness. □

4.8.2 Other Properties

Audit friendliness. `Themis` ' fairness properties are also quite friendly for auditing. In particular, an optimistic fast path can be used where the leader constructs the proposal as normal but does not send any correctness proof to the replicas; instead, the proposal can be validated at a later time, even by an external auditor. This is of course possible only when the auditor can force a revert of the system to an earlier state and/or impose a penalty upon detection of a malicious leader proposal.

Crash-fault tolerant protocol for $n \geq 3f+1$. `Themis` can also be modified in a straightforward way to create an $3f+1$ version (when $\gamma = 1$) in settings with only crash faults (instead of explicit adversarial behavior). Interestingly, the same protocol will work for Byzantine faults but at the cost of allowing the leader node to censor transactions.

4.8.3 SNARK-Themis Details

We now provide details for our SNARK-Themis protocol. Intuitively, for this, instead of forwarding the replica local orderings to all replicas, the leader will now create a SNARK to prove correctness of its proposal.

SNARK preliminaries. For a language L in NP with witness relation $\mathcal{R}_L$, a SNARK for $\mathcal{R}_L$ is a tuple of efficient algorithms (Gen,Prove,Verify) where Gen generates the trusted parameters pp (e.g., the CRS) given 1^λ where λ is the security parameter, Prove is the prover algorithm, and Verify is the verifier algorithm. Given pp and any $(x,w) \in \mathcal{R}_L$, $\mathsf{Prove}(pp,x,w)$ produces a proof π attesting that $x \in L$. The proof can be checked using $\mathsf{Verify}(pp,x,\pi)$. We require the standard completeness, soundness, and knowledge properties for SNARKs. For (asymptotic) efficiency of our protocol, the SNARKs need to have constant-sized proofs excluding a poly-log factor in the security parameter (e.g., [95, 120]). We do not require any zero-knowledge property.

SNARK proof for the leader proposal. Consider the language L in NP such that (x,w) is in the witness relation R_L if the following holds: x is a string representation of directed graph $\mathcal{G}$, w is parsed into $\mathsf{List}_{i_1} \| \cdots \| \mathsf{List}_{i_{n-f}} \| \sigma_{i_1}(\mathsf{List}_{i_1}) \| \cdots \| \sigma_{i_{n-f}}(\mathsf{List}_{i_{n-f}})$ where each $(\mathsf{List}_j, \sigma_j)$ represents a transaction ordering from a distinct node along with a signature from the node, and $\texttt{FairPropose}(\mathcal{L})$ outputs $\mathcal{G}$ where $\mathcal{L} = \{\mathsf{List}_{i_1}, \ldots, \mathsf{List}_{i_{n-f}}\}$. The string representation can be decided upon deterministically, for example, by choosing vertices in alphabetical order. We use general-purpose SNARKS for the language L to prove correctness. A similar correctness proof can be constructed for the update list $\mathcal{E}_{\text{updates}}$.

Finally, the overall proposal block B by the leader will be the tuple $(\mathcal{G},\mathcal{E}_{\text{updates}},\pi=(\pi_1,\pi_2))$ where $\mathcal{G}$ is the graph for newly proposed transactions, $\mathcal{E}_{\text{updates}}$ is the set of update edges, π_1 is the SNARK for $\mathcal{G}$, and π_2 is the SNARK for $\mathcal{E}_{\text{updates}}$. The upshot is that now, the size of B no longer depends on the number of nodes, enabling the underlying consensus algorithm to achieve communication complexity $\mathcal{O}(n)$.

SNARKs for `Wendy` [110] and Aequitas [105]. `Wendy` and Aequitas both require an $\mathcal{O}(n^2)$ communication phase where all nodes gossip transactions to all others. This is critically necessary for censorship resistance, and we found no easy way to improve the communication complexity using SNARKs.

Improved complexity for `Pompē` [168]. The `Pompē` protocol has a communication complexity of $\mathcal{O}(n^2)$. The general design of `Pompē` can be modified to achieve optimistic linear complexity through the use of SNARKs for verifying the computation of the median timestamp. Specifically, after the ordering phase, in `Pompē` , the median timestamp for a given transaction is computed and transmitted during the consensus phase. This is done by the client but `Pompē` assumes that the all clients are protocol nodes. In this case, the median computation can be proved (by the client) using constant-size SNARKs which enables linear communication in the optimistic case.

However, in standard deployments, *clients and protocol nodes can be separate entities* and not all clients need to be present during the system initialization. In such a case, SNARKs with trusted setup might not provide an appropriate trust model for clients. Current constant-size SNARKs, however, all require trusted setup. Further, even if a trusted setup can be used, the public parameters (e.g., the CRS) need to be communicated to clients. Since the CRS size is dependent on n for known constant-size SNARKs, this means that in the standard setting where clients and nodes can be distinct, `Pompē` will not have optimistic communication linear in n.

While SNARKs without trusted setup can be used instead, current state-of-the-art constructions [152, 159, 40] all have at least logarithmic proof sizes. This means that, even with the use of SNARKs, at best, $\mathcal{O}(n\log n)$ optimistic communication complexity can be achieved.

4.9 Experiment Details

We provide experiment details deferred from the main text.

Network ratio and fairness granularity. We highlight that even when the network delay is very large, transaction latencies with smaller r can be approximated by, abstractly, setting a coarser granularity for fairness. A granularity of g (defined below) is equivalent to only quantifying fair ordering across these width g intervals, and ignoring unfairness of transactions within the same interval. Note the time intervals can be w.r.t. the local time at a given node, and synchronized clocks are not required. In other words, the granularity can be changed without affecting any protocol assumptions.

Definition 21 (Fairness granularity)**.** *For granularity g, we consider that timestamps are bucketed into slots of interval g time each (e.g., $[0,g)$, $[g,2g)$ and so on). Events within a time bucket are assumed to happen at the same time.*

4.9.1 Ideal Setting Details

Comparing receive and batch order-fairness. For $n=100$, note the plots for the two definitions are identical except for when $\gamma=0.51$. Even for $\gamma=0.51$, they are identical for a small r. When $n=20$, the two definitions also deviate for $\gamma=0.6$. This is because Condorcet cycles are much more common when r is large and when γn gets close to $n/2$—for large n, this effect is not seen until γ gets close to 0.5.

Abstractly, this deviation can be seen as the result of the frequency with which Condorcet cycles arise in practice. Note that the presence of cycles also means that receive-order-fairness cannot be achieved in these settings. Therefore, a natural question to ask now is how commonly do cycles arise, and when they do, how large do they get? We found that in practical settings, Condorcet cycles are small, and therefore, effectively the two definitions are quite similar.

Origin \ End	us-west-1 (California)	us-east-2 (Ohio)	ap-northeast-1 (Tokyo)	ap-northeast-2 (Seoul)	eu-central-1 (Frankfurt)
us-west-1	-	51.407	105.733	133.771	147.308
us-east-2	51.166	-	131.609	159.931	98.827
ap-northeast-1	106.684	131.359	-	31.723	228.356
ap-northeast-2	134.018	159.561	32.397	-	222.829
eu-central-1	145.962	97.972	228.862	234.641	-

Table 4.1: Average ping times (in ms) between different AWS servers

Cycle size for order-fairness. We find the number of transactions that are present in (strong)-Condorcet cycles of each size in the ideal setting. Fig **??** contains results for $n \in \{20,100\}$ and for reasonable network ratios $r \in \{1,10\}$. For example, for a given length-k cycle, we count k transactions in the bucket for k. Note that $k=1$ means that there is no cycle.

We found that for $r=1$, there were no cycles found for $\gamma \in \{1,0.8,0.6\}$. For $\gamma = 0.51$, for $n=20$, there was one 3-length cycle and one 4-length cycle, while there was just one 3-length cycle for $n=100$.

When $r=10$, we did not find cycles for $\gamma \in \{1,0.8\}$. There were a few length-3 cycles for $\gamma = 0.6$ for $n=20$ but none for $n=100$. For $\gamma = 0.51$, cycles are more common but still a reasonable network ratio of $r=10$, the max cycle length was 11 for $n=20$ and 5 for $n=100$. We also notice that it is less common to find cycles (for the same γ) as n increases due to a smaller variance.

4.9.2 Frontrunning Experiment Details

Remark on full adversarial control. We remark that if the adversary $\mathcal{A}$ has full network control of when transactions (both honest and adversarial) are *input* to honest nodes, then network-level frontrunning is trivial in any protocol. Note that even privacy techniques do not fully prevent this—$\mathcal{A}$ can still perform non-targeted and metadata-based frontrunning. Therefore we explore real-world networks in our analysis instead of ones with full adversarial control.

More specifically, the usual consistency and liveness properties remain intact even if $\mathcal{A}$ has such a full network control. Interestingly, even the fair-ordering property holds based on the input orderings, but since $\mathcal{A}$ can essentially manipulate the input orderings of honest nodes themselves given full network control, the protocol property no longer reflects any intuitive notion of fairness.

Indeed, this limitation led to Kelkar et al. [105] formalizing two networks: the (standard) *internal* network (for communication amongst consensus nodes) and the *external* network (for all transaction submission). The adversary has full control over the internal network but does not control the external network. [105] notes that such a power would be similar in spirit to controlling e.g., the user's access to the internet. This ensures that the input orderings of honest nodes cannot be manipulated by the adversary. We highlight that similar assumptions of the adversary not having full control over transaction submission are present in all previous works [102, 168, 110].

Toy AWS setting. We start with a simple geo-distributed network with one node each in five different AWS regions: us-west-1 (California), us-east-2 (Ohio), ap-northeast-1 (Tokyo), ap-northeast-2 (Seoul), and eu-central-1 (Frankfurt) and measured ping timings between each pair of servers (in both directions) as a proxy for the communication latency between them. Table 4.1 contains ping times averaged over 1000 samples.

Out of 20 total node pairs (A,B), we found that 10 are frontrunnable (i.e., A can frontrun transactions originating from B) in the case of fair separability. As a concrete example based on Table 4.1, an adversary in Tokyo could frontrun a user transaction from Seoul by sending its adversarial transaction to California before the transaction from Seoul reached Frankfurt. On the other hand, there were no instances of triangle inequality being broken in our toy setting, and as a consequence, there were no frontrunnable pairs in the case of order-fairness.

Frontrunning analysis. We now provide an analysis of how `Themis` can handle frontrunning under a common network model where triangle inequality is respected. In a network where triangle

inequality holds, an adversary cannot see an honest transaction and then create and send an adversarial transaction (i.e., an attempt to frontrun) in a way that any honest node receives the adversarial transaction first. In other words, if tx is an honest transaction and m is an adversarial transaction sent afterwards, then all honest nodes should receive tx before m. In this network setting, we can show that `Themis` prevents frontrunning—i.e., the adversarial transaction will be ordered strictly after the honest transaction it is trying to frontrun. We show this through the following lemma.

Lemma 34. *An adversary cannot force an adversarial transaction m to be in the same cycle as any honest transaction under the above mentioned network model.*

Proof. To see why, consider a transaction dependency graph $\mathcal{G}^*$ that contains an edge from tx to tx$'$ is a *majority* of nodes claim to have received tx earlier. First note that this graph is complete (i.e., there is one edge between any transaction pair), and that any cycle in the actual dependency graph will also be a cycle in this graph.

Now, since $\mathcal{G}^*$ is complete, if a cycle within $\mathcal{G}^*$ contains m, then m is also contained within a cycle of length 3, i.e., there exists some x,y such that $x \to y \to m \to x$ is a cycle. Note that this is directly true for `Themis` since its dependency graph is already complete.

Now, if x,y are both honest user transactions, then by our network assumption, all honest users have received y before m and m before x. This implies that no honest node will have received x before y and so the cycle cannot exist. On the other hand, w.l.g, if only x is honest, then again by our network assumption, all honest users must have received m before x and x before y. Therefore, no honest node has received y before m and therefore again the cycle cannot exist. □

This also shows that for `Themis` , and all the order-fairness variants more broadly, assuming triangle inequality holds, an adversary will not be able to get its transaction ordered before the honest user's transaction.

Chapter 5

Garuda : An Approximate Fair Ordering Algorithm

Many protocols have been proposed in the literature with the intention to define more and more stronger notion of fair ordering. However, they all suffer from high communication complexity of at least $\mathcal{O}(n^2)$. This has rendered these protocols expensive to use in practice where n can be in the range of 1000s. Some protocols like `Themis` [103] attempted to solve this issue of communication complexity by using SNARKS in the critical path of the protocol. However, current state-of-the-art SNARKs have high proving time, thus, lending to high finalization time.

In this work, we propose a new order-fairness primitive called as (γ,μ)-*approximate order-fairness*: informally, if there is are sufficient number of honest replicas which received tx_1 at least 2×network delay before tx_2, then tx_1 must be ordered before tx_2. We propose a protocol called `Garuda` that guarantees (γ,μ)-*approximate order-fairness* while incurring $\mathcal{O}(n)$ communication complexity without using any SNARKS in the critical path. This makes the protocol very practical and scalable.

5.1 Introduction

Decentralized Finance (DeFi), meaning the deployment of financial instruments on blockchains, has attracted substantial interest in recent years, with over 54 billion USD locked in DeFi protocols as of April 2023. Unfortunately, while DeFi continues to gain popularity, a long line of work [64, 76, 141, 171] has shown the rise of adversaries extracting profit by manipulating the ordering and inclusion of transactions in DeFi applications. In decentralized exchanges and lending contracts, for example, where transaction execution order is critically important, such order manipulation results in attackers profiting at the expense of ordinary users.

Order manipulation is possible in existing protocols largely because the formal properties required of state machine replication (SMR) or consensus—the primitive that underpins blockchains— place no restriction on how transactions are ordered. Neither consistency nor liveness, the two pillars of consensus security, enforces any relationship between the order in which transactions arrive in the network and their final ordering. Indeed, in both permissioned consensus protocols, e.g., PBFT [50] and Hotstuff [163], and permissionless ones, e.g., Ethereum, the current "leader" fully controls the inclusion and ordering of transactions within a block that it creates.

To address this gap in traditional consensus research, a recent line of work [45, 105, 106, 110, 168] has proposed protocols with socalled fair ordering properties—properties that prevent adversarial manipulation of transaction ordering. These works propose several definitions of fairness along with protocols that realize them. Intuitively, this style of fairness seeks to guarantee a specific ordering in the finalized ledger based on how transactions arrive into the network.

The fair ordering landscape. Existing fair-ordering protocols have come in many flavors. Some provide strong fairness guarantees such as batch-ordering [106, 105, 103] where the guarantee is that if γ ($\frac{1}{2} < \gamma \leq 1$) of replicas received tx_1 before tx_2, then tx_1 should be ordered no later than tx_2. Presence of non-transitive Condorcet cycles in message receipt times across replicas gives rise to batches of transactions among which no strict ordering can be guaranteed. However these cycles can extend arbitrarily long. Furthermore, `Themis` [103], which is the only practical implementation to guarantee batch-ordering, requires $\mathcal{O}(n^2)$ communication complexity for any proposed block of transactions to be committed even in optimistic path. The authors suggest a variant of Themis where the leader runs a zero-knowledge (ZK) prover to generate a proof for correctness of finalization algorithm while achieving $\mathcal{O}(n)$ communication complexity. However, proof generation every time a new block has to be proposed by the leader puts a lot of negative externality in terms of latency and throughput even in the optimistic path. For some numbers, generating a ZK proof for hash chain with common hash functions like SHA256 or BLAKE3 requires seconds in some of the state-of-the-art ZK proving systems [18]. Naturally, given more complexity, one can expect generating ZK proof for correct execution of `Aequitas` finalization algorithm by the leader in `Themis` would require even more time.

Some provide weaker fairness guarantees such as ordering-linearizability [168] and timed-relative-fairness [110]. Time-related-fairness basically guarantees that if all honest replicas received tx_1 before tx_2, then tx_1 will be ordered before tx_2 whereas ordering-linearizability applies the same guarantee only on the transactions that are ordered in the ledger eventually. However, even with weaker fairness guarantee of timed-relative-fairness, `Wendy` [110] requires each replica to broadcast local timestamp of each transaction to all other replicas, thus, incurring $\mathcal{O}(n^2)$ communication complexity. Similarly, `Pompē` [168] requires leader to collect time stamps associated with each transaction from $\mathcal{O}(n)$ replicas and broadcast all of it back to all replicas, which incurs $\mathcal{O}(n^2)$ communication complexity.

5.1.1 Contributions and overview

In the work, our goal is to develop an ordering protocol with $\mathcal{O}(n)$ communication complexity that achieves the reasonable fairness guarantee. Towards that end, we define γ-*approximate ordering*: if at least γ honest replicas receive tx_1 at least 2δ secs before they receive tx_2, then tx_1 is ordered before tx_2. There are two unknowns here: what should be the values of γ and δ. γ is chosen to be the minimum integer such that every honest replica can be convinced with optimistic responsiveness that there exists at least one honest replica who has received tx_1 at least 2δ secs before they receive tx_2. We show that $\gamma = 2f + 1$. As for δ, if it is equal to real network delay δ_{real}, then given the antecedent, all honest replicas must have received tx_1 before tx_2 and so tx_1 should be ordered before tx_2. However, partial synchrony assumption states that δ_{real} is unknown. One way to circumvent this issue is by setting δ to be equal to a rough estimate δ_{est} of real network delay δ_{real}. This estimate δ_{est} is maintained as a common knowledge across all honest replicas and is updated to be as close to δ_{real} as possible.

Given that δ_{est} is maintained as a common knowledge, it needs to be set at a value that will be acceptable to most honest replicas. A very natural choice is to have δ_{est} set at a value such that end-to-end network delay in μ percentile of all messaging in the peer-to-peer network is less than δ_{est} (generally referred to as Pμ). Many data sets pertaining to measurement of netowrk latency are publicly available and are constantly updated [**?**]. Furthermore, previous works have shown that cumulative distribution function (CDF) around network latency don't change much with time [98]. Although `Garuda` features a sub-protocol for upgrade of common knowledge of δ_{est}, rarely any upgrade will be necessary if some additional slack is incorporated whenever an upgrade to common knowledge of δ_{est} is made.

Observe that γ-approximate ordering is a weaker ordering primitive than time-related-fairness. This is because the antecedent of γ-approximate ordering requires at least γ honest replicas to have tx_1 being received at least δ_{est} secs before tx_2 which in turn, assuming $\delta_{est} (\approx \delta_{real})$, implies that tx_1 is received before tx_2 in all honest replicas. On the other hand, antecedent for time-related-fairness simply requires tx_1 is received before tx_2 in all honest replicas. However, unlike `Wendy` or `Pompē`, we construct an ordering protocol `Garuda` that achieves $2f+1$-approximate ordering with $\mathcal{O}(n)$ communication complexity in the optimistic path.

From the point of implementation, `Garuda` has been designed as an adapter on any existing BFT protocol. That is, both protocols feature an merging algorithm for leader and additional validity

Protocol	Transaction Ordering	Comm. Complexity Optimistic	Comm. Complexity Worst	ZK proofs in optimistic path	Corruption	Liveness
Aequitas [105]	γ-batch-order-fairness	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	No	$n > \frac{4f}{2\gamma-1}$	Weak
`Wendy` [110]	Timed-Relative-Fairness	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	No	$n \geq 3f+1$	Standard
`Pompē` [168]	Ordering Linearizability	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	No	$n \geq 3f+1$	Standard
`Themis` [103]	γ-batch-order-fairness	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	No	$n > \frac{4f}{2\gamma-1}$	Standard
SNARK-Themis [103]		$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	Yes		
`Garuda` (This work)	(γ,μ)-approximate-order-fairness	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	No	$n > \max\left(5f+2(1-\mu)n+1, \frac{4f}{2\mu-1}\right)$	Standard

Table 5.1: Comparison of different order-fairness protocols.

conditions for replicas as part of underlying BFT consensus.

Table 5.1 illustrates some of the comparison points between `Garuda` and existing fair-ordering protocols.

5.2 Model

We consider the system to be composed of a fixed set of n replicas, indexed by $i \in [n]$ where $[n] = \{1,...,n\}$.

Corruption model. We consider byzantine fault tolerant model wherein a replica is said to be honest in a given view if it is never under adversarial control, otherwise it is said to be byzantine. We assume that once a replica is corrupted, it cannot become honest at a later point. In our general model, we assume that the adversary can corrupt replicas dynamically. That is, replicas can be corrupted at any point during the protocol's execution. A set $F \subset [n]$ of up to f replicas are Byzantine, and remaining ones are honest. We will often refer to the Byzantine replicas as being coordinated by one single adversary, which learns all internal aspect of these replicas.

Adapter. We assume that the n replicas are participating in an underlying BFT consensus protocol. Our goal is to design an adapter on top of this BFT consensus protocol that enables the whole system to guarantee $\gamma-$linear approximate order fairness. This adapter takes the shape of (1) data that replicas have to send to the leader of each view, (2) pre-processing on the data by the leader leader in order come up with the proposal for that view, and (3) additional validity conditions that replicas need to run on top of underlying BFT protocol's validity conditions.

Network. We formally define the different network assumptions for both the external and internal networks. The external network models the communication network between clients who are using the system by sending transactions and the replicas that are participating in the BFT consensus protocol. We assume that this external network is a peer-to-peer (P2P) network with an unknown network delay δ_{real} secs, that is, any transaction sent by a client in the external network is guaranteed to reach all other nodes in the external network within a bounded delay δ_{real}. In this work, we abstract out the internal details of P2P network and won't bother with how replicas behave in this network. For the external network, let us assume δ_{est} to be such that end-to-end network delay from μ percentile of all replicas is less than δ_{est}.

The internal network represents the communication network among replicas for the purpose of message exchanges as defined by the underlying BFT protocol. This internal network is assumed to be partial asynchronous with a known bound Δ on message delay after some unknown global stablization time (GST).

Notation. For any two transactions (tx_1,tx_2), $tx_1 \prec_{(m,\delta)} tx_2$ represents that the timestamp for

tx_1 is more than δ secs before that of tx_2 in at least m replicas. For any replica R_i, $\mathcal{L}i$ consists of transactions along with the timestamps at which those transactions were received by the replica R_i locally. We denote the timestamp of a transaction tx recorded in the replica R_i as $\texttt{TS}(tx,i)$.

5.3 Approximate Order-Fairness

In practical blockchain deployments, network delay δ_{real} of the underlying P2P network give rises to a very natural definition of *fairness*: if an honest replica has estimated delay to be δ_{real} secs with high confidence and if transaction tx_1 was received at least $2\delta_{real}$ secs before transaction tx_2, then the replica can expect that no other replica has received tx_2 before tx_1. From the point of view of the honest replica, it should be totally fair to have tx_1 ordered before tx_2 in the ledger.

However, it could be the case that only one replica received tx_1 at least $2\delta_{real}$ secs before transaction tx_2 while others did receive tx_1 before transaction tx_2 but not more than $2\delta_{real}$ secs before. Considering many of the replicas to be byzantine in nature, it is essential that sufficient number of replicas have received tx_1 at least $2\delta_{real}$ secs before transaction tx_2 such that remaining honest replicas are assured that at least one honest replica received tx_1 at least $2\delta_{real}$ secs before transaction tx_2. We capture this in the following primitive:

$\gamma-$**Linear approximate order-fairness.** Linear approximate-order fairness states that if $\gamma-$number of honest replicas received transaction tx_1, that has not been committed yet, sufficiently earlier than another transaction tx_2 (could be from different clients), then tx_1 should be ordered before tx_2 in the final ledger by every honest replica and the whole process of coming to this consensus should only incur $\mathcal{O}(n)$ communication complexity.

Requirement for linear communication complexity can only be achieved by using a leader-based protocol, with the leader serving as a relay for communication between replicas. An immediate question is how large should be the separation between transactions tx_1 and tx_2 so that all honest replicas can come to the same consensus on the ordering between them. Clearly, if an honest replica received transaction tx_1 more than $2\delta_{real}$ secs before tx_2, then it is guaranteed that all other honest replicas must have received transaction tx_1 before tx_2. However, there are two major issues with this observation: (1) the leader can be byzantine and therefore, the information on transaction timestamps sent by the honest replica who has received transaction tx_1 more than $2\delta_{real}$ secs before tx_2 can be conveniently dropped by the byzantine leader without any accountability and won't be received by other honest replicas, (2) under partial synchrony model, no honest replica has knowledge about the actual network delay δ_{real} and different honest replicas might have different estimates of δ_{real}.

In order to circumvent the lack of knowledge of δ_{real}, all honest replicas instead maintains a consistent estimate δ_{est} of δ_{real}, that is, each honest replica has the same estimate on δ_{real}. Obvious question is what should be δ_{est} set at so that it is acceptable to most honest replicas. Given that δ_{real} is unknown and typically we can determine the CDF of the sampling distribution of δ_{real} by network measurements [98, 158], a system parameter μ can be set at genesis such that at least $P\mu$ of the CDF of sampling distribution of δ_{real} is less than estimate δ_{est} (changing μ would require hard fork). Hence, we redefine $\gamma-$linear approximate order fairness as:

$(\gamma,\mu)-$**linear approximate order fairness.** If $\gamma-$number of honest replicas received transaction tx_1, that has not been committed yet, at least $2\delta_{est}$ secs earlier than another transaction tx_2 (could be from different clients), then tx_1 should be ordered before tx_2 in the final ledger by every honest replica and the whole process of coming to this consensus should only incur $\mathcal{O}(n)$ communication complexity.

Here, δ_{est} is a function of μ.

We next show that if at most $2f$ honest replicas receive transaction tx_1 more than $2\delta_{est}$ secs before tx_2, then byzantine replicas can convince honest replicas to come to consensus on tx_2 being ordered after tx_1 in the final ledger for any leader-based protocol with linear communication.

Lemma 35. *For any $\gamma <= 2f + (1-\mu)n$, there exists no protocol that can achieve $(\gamma, \mu)-$linear approximate order-fairness.*

Proof. Suppose we want to guarantee that if even one honest replica has received tx_1 more than $2\delta_{est}$ secs before tx_2, then tx_2 shouldn't be ordered before tx_1 in the final ordering. Assuming we want to ensure liveness, the leader will wait for response from at least $n-f$. However, it could be the case that response from that one honest replica containing time stamps of tx_1 and tx_2 might not be received by the leader due to the partial asynchronous nature of our protocol. So, there is no way to guarantee this property. Suppose we relax the guarantee to that if at least $f+1$ honest replica has received tx_1 more than $2\delta_{est}$ secs before tx_2, then tx_2 shouldn't be ordered before tx_1 in the final ordering. Then the dilemma is leader cannot distinguish between these two scenarios: (1) response from only one of this honest replica are received by the leader and, (2) response from an adversarial replica is received that shows tx_1 received than $2\delta_{est}$ secs before tx_2 (this timestamp was tampered maliciously). Additionally, it could be the case that $(1-\mu)n$ honest replicas are facing end-to-end network delays in the external network that exceed δ_{est} but their response got included by the leader (internal network is assumed to be separate, see sec 5.2). □

`Garuda` achieves optimal result by guaranteeing that if at least $2f+(1-\mu)n+1$ honest replicas receive transaction tx_1 more than $2\delta_{est}$ secs before tx_2, then tx_1 will be ordered before tx_2.

The requirement for having all honest replicas maintain a consistent estimate δ_{est} necessitates that δ_{est} be acceptable for most honest replicas while taking into consideration the long-tailed nature of sampling distribution of δ_{real}. Given that these distributions change (although sporadically), it is necessary that the δ_{est} is updated consistently to a new estimate across all honest replicas. To capture this requirement, we define the primitive of upward mobility. Upward mobility ensures that if the assumption that at least μ percentile of end-to-end network delay across all messaging is less than δ_{est} is no longer valid, then δ_{est} should get updated across all honest replicas.

$(\chi_1,\chi_2,\chi_3)-$**upward mobility.** Upward mobility states that

1. if there exists even one pair of transactions tx_1 and tx_2 such that χ_1 number of honest replicas have received transaction tx_1 more than $2\delta_{est}$ secs before tx_2 and χ_2 number of distinct honest replicas have received transaction tx_2 more than $2\delta_{est}$ secs before tx_1, then a leader must be forced to trigger an update such that δ_{est} is increased across all honest replicas.

2. if there exists no pair of transactions tx_1 and tx_2 such that sufficient number of honest replicas have received transaction tx_1 more than $2\delta_{est}$ secs before tx_2 and χ_3 number of distinct honest replicas have received transaction tx_2 before tx_1, then a byzantine leader shouldn't be able to trigger an update to δ_{est} at even one honest replica.

Part (1) of the definition states a condition beyond which an adversarial leader cannot prevent an update to δ_{est} across all honest replicas. Part (2) of the definition states a condition below which an adversarial leader will not be able to trigger an update to δ_{est} across any honest replica. χ_1,χ_2,χ_3 should be set such that every honest replica can be assured that there exists at least one honest replica whose previous observation of network delay from itself to anyone else in the external network being less than δ_{est} is no longer satisfied.

Recall that δ_{est} is defined to be such that at least μ percentile of end-to-end delay across all message connections are less than δ_{est}. It can happen that δ_{est} is such that it is much higher than Pμ in the CDF of the sampling distribution of δ_{real}. In such a case, it might be nice to reduce δ_{est} to a value that is much closer Pμ in the CDF of the sampling distribution of δ_{real}. CDF for sampling distribution of δ_{real} is typically long-tailed and having δ_{est} close to $P\mu$ of this CDF gives tighter quality of service in terms of approximate order-fairness to clients. This is captured in our following definition of downward mobility. We would like to point out that downward mobility is not a necessity as the assumption on δ_{est} being greater than Pμ of the CDF of δ_{real} is still satisfied. However, we will still explore the primitive of downward primitive from the point of improving the quality of service and later in Sec. 5.4, we will describe how `Garuda` achieves it.

Given byzantine nature of the system, an important question is how can each replica be convinced that δ_{est} is far from $P\mu$ of the CDF of δ_{real} and that every honest replica knows about it. If we consider that local clocks of all honest replicas are synchronized, then in an leader-based protocol, an honest leader can just use the spread of time stamps corresponding to each transaction across many replicas to check whether δ_{est} needs to be reduced or not. However, with no assumption of synchronized local clocks of honest replicas, we instead have to rely on spread of relative separation of time stamps of each pair of transactions across many replicas. This is captured in the following primitive:

$\tau-$downward mobility. Consider $\delta_{new-est} < \delta_{est}$ but $\delta_{new-est}$ is greater than $P\mu$ of the CDF of δ_{real}. Strong downward mobility states that for each pair of transactions (tx_1, tx_2) that have not yet been committed, if at least τ honest replicas have $\texttt{TS}(tx_2, i) - \texttt{TS}(tx_1, i) \in [t, t + 2\delta_{new-est}]$, for some t, then an honest replica must be able to trigger an update such that δ_{est} is decreased to $\delta_{new-est}$ across all honest replicas. On the other hand, if at least τ honest replicas don't have $\texttt{TS}(tx_2,i)-\texttt{TS}(tx_1,i)\in[t,t+2\delta_{new-est}]$, for some t, then a leader must not be able to trigger an update to δ_{est}.

Both upward and downward mobility are defined with the intention of getting $\delta_{new-est}$ to be as close as possible to $P\mu$ of the CDF of sampling distribution of δ_{real}. However, given that a byzantine leader is not accountable if it doesn't trigger downward mobility of δ_{est} even though it should be, what can happen is that $\delta_{new-est}$ remains substantially larger than $P\mu$ of the CDF of sampling distribution of δ_{real} until an honest replica is selected as part of the leader-rotation of the underlying BFT protocol. This can impact the quality of service.

5.4 Protocol

5.4.1 Protocol description

The pseudocode for alhorithm that the leader and each replica has to run in `Garuda` is given in Algorithm 3 and 4, respectively.

At the beginning of each view of the underlying BFT consensus protocol, each honest replica R_i collects all transactions in its local mempool that are not yet committed in the ledger in $\texttt{List}_i$ and send it to the leader in that view. The leader waits until it receives these `List` from at least $n - f$ replicas. Let $\mathcal{C}$ denote the collection of all such $n-f$ lists.

Upward Mobility. On receiving `List` from $n-f$ replicas, the leader determines if there are any pairs of transactions (tx_1,tx_2) such that both haven't been committed yet but $tx_1 \prec_{(f+1,2\delta_{est})} tx_2$ and $tx_2 \prec_{(f+1,0)} tx_1$. We denote this set by $\mathcal{T}$. If there is even one pair of transaction (that is, $\mathcal{T} \neq \phi$), then it signals that δ_{est} needs to be increased. The leader now has to find the appropriate value to which δ_{est} has to be updated. Towards that end, the leader finds the minimum possible value $\delta_{new-est}$ which ensures that for all pair of transactions $tx_1, tx_2 \in \mathcal{T}$, we have $tx_1 \not\prec_{(f+1,2\delta_{new-est})} tx_2$. The leader then collects all the appropriate `List` as evidence $\mathcal{E}_{(tx_1,tx_2)}$, the proposed $\delta_{new-est}$ and send the proposal $< \texttt{upward}, (tx_1,tx_2), \mathcal{E}_{(tx_1,tx_2)}, \delta_{new-est} >$ to the replicas as part of the BFT consensus protocol. The replicas, on receiving this message, use the evidence $\mathcal{E}$ to check whether update of δ_{est} is necessary or not and then also verifies whether the proposed $\delta_{new-est}$ suffices or not. This serves as an additional validity condition on top of existing validity conditions that exists in BFT consensus protocol.

Downward Mobility. Next, the leader checks whether δ_{est} needs to be decreased. Towards that end, the leader first determines the set $\mathcal{U}$ of transactions that are present in `List` from at least $\mu n - 2f$ replicas. Then it determines all those transactions $tx \in \mathcal{C}$ such that they are included in at least $\mu n - 2f$ replicas. Now, if the leader is able to determine a $\delta_{new-est} < \delta_{est}$ such that for all $tx_1, tx_2 \in \mathcal{U}$, if $|\texttt{TS}(tx_2, i) - \texttt{TS}(tx_1, i)|$ from at least $\mu n - 3f$ replicas R_i lie within some

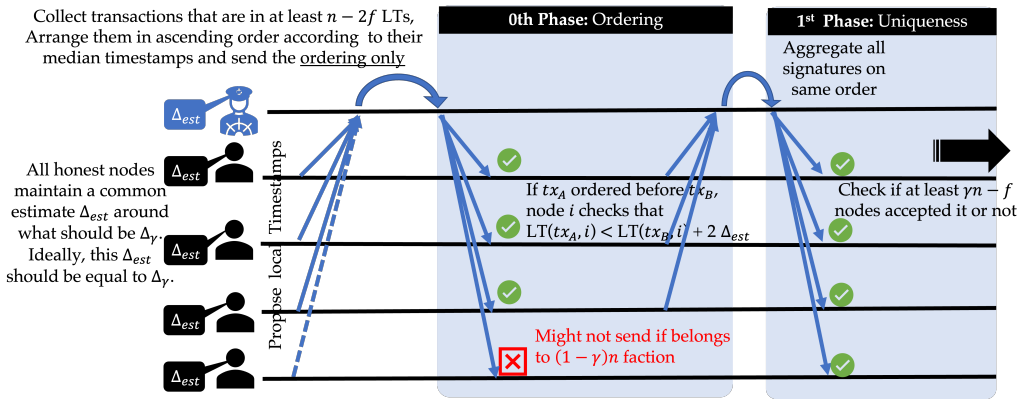


Figure 5-1: Approximate ordering in `Garuda` . Here LT refers to a table of local timestamps associated with different transactions.

interval $[t, t + 2\delta_{new-est}]$ for some t, then the leader can trigger an update of δ_{est} in all honest replicas. To do that, the leader generates evidence $\mathcal{E}_{(tx_1,tx_2)}$ that consists of reference pointers in $\mathcal{C}$ and associated value of time t. We require at least $\mu n - 3f$ replicas because for all $tx_1, tx_2 \in \mathcal{U}$, at least $\mu n - 3f$ replicas will contain both tx_1, tx_2. The leader then sends the proposal $<$ `downward`$, \mathcal{C}, \bigcup_{tx_1,tx_2} \mathcal{E}_{(tx_1,tx_2)}, \delta_{new-est} >$ to the replicas as part of the BFT consensus protocol. The replicas, on receiving this message, use the evidence $\mathcal{E}_{(tx_1,tx_2)}$ to check whether update of δ_{est} is necessary or not and whether the proposed $\delta_{new-est}$ suffices or not. This serves as an additional validity condition on top of existing validity conditions that exists in BFT consensus protocol.

Approximate ordering. If neither upward mobility or downward mobility is triggered, then the leader takes the normal route of proposing an ordering of new set of uncommitted transactions. For each of the transactions in the set $\mathcal{U}$, their respective median is computed (by using the timestamps available in the `List` from at least $\mu n - 2f$ replicas) and are then arranged in ascending order based on the computed median. This is denoted by the `B`. The leader then proposes the message $<$ *order*, `B` $>$ as part of the consensus protocol in that view. On receiving $<$ *order*, `B` $>$ from the leader, the replica R_i checks that for all pairs of transactions $tx_1, tx_2 \in$ `B` such that $tx_1 \rightarrow tx_2$ or if tx_1 is present but tx_2 is not, check that $\texttt{TS}(tx_1) < \texttt{TS}(tx_2) + 2\delta_{est}$. This serves as an additional validity condition on top of existing validity conditions that exists in BFT consensus protocol.

5.5 Security

Lemma 36. *Consider β to be the adversarial threshold tolerable for the underlying consensus protocol. For $n > \frac{4f}{2\mu-1}$, `Garuda` satisfies $(2f+1+(1-\mu)n, \mu)$−linear approximate order-fairness.*

Proof. First we show completeness. Observe that if $\texttt{TS}(tx_2)$ is more than $\texttt{TS}(tx_1)$ by at least $2\delta_{est}$ secs in signed lists from at least $2f+1+(1-\mu)n$ honest replicas, this would imply that there are at least $\mu n - f$ honest replicas for whom $\texttt{TS}(tx_2)$ is more than $\texttt{TS}(tx_1)$. $\mathcal{L}$ from at least $\mu n - 2f$ honest replicas would be received by the leader. Given $\mu n - 2f > \frac{n}{2}$, we have median(tx_1) < median(tx_2). This also implies that at least $\mu n - f$ honest replicas agrees with the leader's proposed ordering between tx_1 and tx_2 when they receive $<$`order`,`B`$>$ from the leader. Therefore, these $\mu n - f$ honest replicas will respond by sending back $<$`preaccept`,`hash`(`B`)$>$ to the leader. The leader will then be able to aggregate signature from at least $\mu n - f$ replicas, which will be accepted by each of the $n - f$ honest replicas.

Now, we check for soundness. Consider any pair of transactions (tx_1, tx_2) such that tx_2 is ordered before tx_1 in the final ordering by the leader but $\texttt{TS}(tx_2)$ is more than $\texttt{TS}(tx_1)$ by at least $2\delta_{est}$ secs in

Algorithm 3 `Garuda` : Leader role

1: **In view v:**

1. Wait until correctly signed lists `List` are received from $n-f$ replicas. Let $\mathcal{C}$ denote the collection of all such $n-f$ lists.
2. Determine $\mathcal{T} := \{(tx_1,tx_2) \mid tx_1 \prec_{(f+1,2\delta_{est})} tx_2 \wedge tx_2 \prec_{(f+1,0)} tx_1 \wedge tx_1,tx_2 \notin \mathcal{L}\}$. Note that it could be the case that $(tx_1,tx_2) \in \mathcal{T}$ but $(tx_2,tx_1) \notin \mathcal{T}$. If $\mathcal{T} \neq \phi$, then
   (a) $\forall \quad tx_1,tx_2 \in \mathcal{T}$, collect all such lists $\texttt{List}_i$ that can serve as the evidence for $(tx_1,tx_2) \in \mathcal{T}$. Let this collection of lists be denoted by $\mathcal{E}_{(tx_1,tx_2)}$.
   (b) $\forall \quad (tx_1,tx_2) \in \mathcal{T}$, determine $\delta_{new}^{(tx_1,tx_2)}$ such that $tx_1 \prec_{\left(f+1,2\delta_{new}^{(tx_1,tx_2)}\right)} tx_2 \wedge tx_2 \prec_{(f+1,0)} tx_1 = 0$. Let $\delta_{new-est} = 2 \times \min_{\mathbb{R}^+ \cup \{0\}} \max_{(tx_1,tx_2) \in \mathcal{T}} \delta_{new-est}^{(tx_1,tx_2)}$ and $(tx_1^\star,tx_2^\star) = \text{argmax}_{(tx_1,tx_2) \in \mathcal{T}} \delta_{new-est}^{(tx_1,tx_2)}$.
   (c) Run the underlying consensus protocol to commit to the valid update on the proposal $<\texttt{upward},(tx_1^\star,tx_2^\star),\mathcal{E}_{(tx_1^\star,tx_2^\star)},\delta_{new-est}>$.
   (d) Exit to next view $v+1$ with leader in the current view v continue to be the leader in the next view.
3. For each unique tx appearing in $\mathcal{C}$, determine $\mathcal{S}_{tx} = \{\texttt{List}_i | \texttt{List}_i \in \mathcal{C} \wedge tx \in \texttt{List}_i\}$. Construct the set $\mathcal{U} = \{tx | |\mathcal{S}_{tx}| \geq n-2f\}$. If $|\mathcal{U}| \neq \phi$, then
   (a) $\forall tx_1,tx_2 \in \mathcal{U}$, determine $|\texttt{TS}(tx_2) - \texttt{TS}(tx_1)|$ from the `List` of at least $\mu n - 3f$ replicas. We represent these data points as $\mathcal{V}_{(tx_1,tx_2)}$.
   (b) Determine whether there exists $\delta_{new-est} < \delta_{est}$ such that $\forall tx_1,tx_2 \in \mathcal{U}$, all $\mu n - 4f$ of the data points in $\mathcal{V}_{(tx_1,tx_2)}$ are within an interval $[t,t+2\delta_{new-est}]$ for some t. If yes, then collect the reference pointers to all the associated set of `List` along with associated t as $\mathcal{E}_{(tx_1,tx_2)}$. If no, continue to Step 4.
   (c) Run the underlying consensus protocol to commit to the valid update on the proposal $<\texttt{downward},\mathcal{C},\bigcup_{tx_1,tx_2 \in \mathcal{U}} \mathcal{E}_{(tx_1,tx_2)},\delta_{new-est}>$.
   (d) Exit to next view $v+1$ with leader in the current view v continue to be the leader in the next view.
4. If none of the either case, then
   (a) Arrange all transactions in $\mathcal{U}$ in ascending order of median(tx). Let `B` denotes this ordered list of transactions.
   (b) Send the proposal $<\texttt{order},\texttt{B}>$ to all replicas.
5. On receiving message $<\texttt{preaccept},\sigma_i(\texttt{hash}(\texttt{B}))>$ from at least $\mu n - f$ replicas, aggregate all the signatures $\sigma_i(\texttt{hash}(\texttt{B}))$ into $\sigma_{agg}(\texttt{hash}(\texttt{B}))$. Send the message $<\texttt{aggregate},\sigma_{agg}(\texttt{hash}(\texttt{B}))>$ to all replicas as part of the underlying consensus protocol after the confirmation, append `B` to the list of agreed upon blocks.

at least $2f+1+(1-\mu)n$ honest replicas. That would also mean at least $\mu n-2f$ honest replicas agreed with this proposed ordering. However, by assumption, at least $2f+1+(1-\mu)n$ honest replicas won't sign in this proposed ordering as it doesn't comply with their local verification check. This leads to contradiction. □

Lemma 37. *`Garuda` satisfies $(2f+1+(1-\mu)n,2f+1+(1-\mu)n,2f+1+(1-\mu)n)$-upward mobility.*

Proof. First we show completeness. Suppose the leader collected signed lists from at least $f+1+(1-\mu)n$ replicas that have $\texttt{TS}(tx_1)$ before $\texttt{TS}(tx_2)$ and signed lists from at least $f+1+(1-\mu)n$ replicas that have $\texttt{TS}(tx_2)$ before $\texttt{TS}(tx_1)$ by more than $2\delta_{est}$ secs. This evidence that the leader sends is enough to convince all other honest replicas that there CDF of δ_{real} has shifted such that $P\mu$ of the CDF of δ_{est} is no longer within δ_{est}. Therefore, all honest replicas will agree to an honest leader's proposal to increase δ_{est} to $\delta_{new-est}$.

Next, we show soundness. It has two parts:

- Suppose there is a transaction pair (tx_1,tx_2) such that at least $2f+1+(1-\mu)n$ honest replicas have received transaction tx_2 more than $2\delta_{est}$ secs before tx_1 and there are at least $2f+1+(1-\mu)n$ distinct honest replicas that have received transaction $tx1$ more than $2\delta_{est}$ secs before transaction tx_2. However, if the malicious leader, instead of proposing an `upward` message, pursues with `order` message (with any order among these two transactions), then it will not receive attestation from at least $2f+1+(1-\mu)n$ replicas.

- Suppose there are no transaction pairs among the lists received in the current view such that at least $2f+1+(1-\mu)n$ honest replicas have received transaction tx_2 before tx_1 and there are at least $2f+1+(1-\mu)n$ distinct honest replicas that have received transaction $tx1$ more than $2\delta_{est}$

Algorithm 4 `Garuda` : Replica role

1: **In view v, at the replica R_i :**

1. Let $\texttt{List}_i$ denote the set of transactions that were received by the replica R_i along with their local timestamps, that is not yet part of the ledger of committed transactions $\mathcal{L}$.
2. Send $\texttt{List}_i$ to the leader of the current view v.
3. If the leader responds with the proposal $<\texttt{upward},(tx_1,tx_2),\mathcal{E}_{(tx_1,tx_2)},\delta_{new-est}>$, then
   (a) Use $\mathcal{E}_{(tx_1,tx_2)}$ to verify that $tx_1 \prec_{(f+1,2\delta_{est})} tx_2 \wedge tx_2 \prec_{(f+1,0)} tx_1 \wedge tx_1,tx_2 \notin \mathcal{L} = 1$.
   (b) Verify that the proposed $\delta_{new-est}$ is the minimum possible value that results in $tx_1 \not\prec_{(f+1,2\delta_{new-est})} tx_2 \wedge tx_2 \prec_{(f+1,0)} tx_1 = 0$.
   (c) Participate in the underlying consensus protocol to commit to the proposed update to δ_{est} from the leader.
   (d) On commitment of the proposal $<\texttt{upward},(tx_1,tx_2),\mathcal{E}_{(tx_1,tx_2)},\delta_{new-est}>$, update δ_{est} to $\delta_{new-est}$.
4. If the leader responds with the proposal $<\texttt{downward},\mathcal{C},\bigcup_{tx_1,tx_2\in\mathcal{U}}\mathcal{E}_{(tx_1,tx_2)},\delta_{new-est}>$, then
   (a) Construct the set $\mathcal{U}$ and $\mathcal{V}_{(tx_1,tx_2)} \forall tx_1,tx_2 \in \mathcal{U}$.
   (b) Use $\delta_{new-est}$ and $\bigcup_{tx_1,tx_2\in\mathcal{U}}\mathcal{E}_{(tx_1,tx_2)}$ to verify that $\forall tx_1,tx_2 \in \mathcal{U}$, at least $\mu n - 4f$ of the data points in $\mathcal{V}_{(tx_1,tx_2)}$ are within an interval $[t,t+2\delta_{new-est}]$ for some t.
   (c) Participate in the underlying consensus protocol to commit to the proposed update to δ_{est} from the leader and on commitment of the proposal, update δ_{est} to $\delta_{new-est}$.
5. If the leader responds with the proposal $<\texttt{order},\texttt{B}>$, then
   (a) For all transaction pair (tx_1,tx_2) that is ordered $tx_1 \rightarrow tx_2$ in the $\texttt{B}$ or if tx_1 is present but tx_2 is not, check that $\texttt{TS}(tx_1) < \texttt{TS}(tx_2) + 2\delta_{est}$.
   (b) If yes, continue participating in the underlying consensus protocol.
6. On receiving message $<\texttt{aggregate},\sigma_{agg}(\texttt{hash}(\texttt{B}))>$ from the leader, check that signature from at least $\mu n - f$ replicas are included in $\sigma_{agg}(\texttt{hash}(\texttt{B}))$. If yes, continue participating in the underlying consensus protocol and after the confirmation, append $\texttt{B}$ to the list of agreed upon blocks.

secs before transaction tx_2. Now, the malicious leader won't have sufficient evidence to convince at least $\mu n - 2f$ honest replicas that δ_{est} needs to be updated.

Thus, `Garuda` achieves upward mobility. □

Lemma 38. *`Garuda` achieves downward mobility.*

Proof. We start by making the observation that adversarial replicas have no benefit from having downward mobility. Instead having downward mobility makes it harder for adversarial replicas to violate approximate order-fairness. This is because if $\delta_{real} < \delta_{est}$, then an adversary is able to manipulate the transaction ordering so that tx_2 is ordered before tx_1 without violating linear approximate order-fairness even if all honest replicas received tx_1 at least $2\delta_{real}$ secs before tx_2. Therefore, we only show completeness. When the evidence $\mathcal{E}_{(tx_1,tx_2)}$ shows that for the pair of transactions (tx_1,tx_2), $\mathcal{L}$ from at least $n-3f$ replicas have timestamps within the interval $[t,t+\delta_{new-est}]$, then all these replicas are honest. Given that we have $n \geq 6f+1$, this ensures that at least $\frac{n}{2}$ honest replicas agree to decreasing δ_{est} to $\delta_{new-est}$. Thus, `Garuda` achieves downward mobility. □

If we want to remove the assumption that adversarial replicas have no benefit from downward mobility, then we need to have $n \geq 8f+1$ in order to ensure that at least $n-4f$ honest replicas have received transactions within some interval $[t,t+\delta_{new-est}]$. With $n \geq 8f+1$, we have $n-4f \geq \frac{n}{2}$.

Theorem 15. *`Garuda` guarantees safety.*

Proof. Lemma 36, 37 and 38 show that `Garuda` guarantees safety. □

Theorem 16. *`Garuda` guarantees liveness.*

Proof. `Garuda` has been designed as an adapter on top of any existing BFT protocol and it is ensured that all honest replicas always maintain the same δ_{est}. This ensures liveness in `Garuda` . □

5.6 Conclusion

In this work, we propose a new order-fairness primitive called as (γ,μ)-*approximate order-fairness*: informally, if there is are sufficient number of honest replicas which received tx_1 at least 2×network delay before tx_2, then tx_1 must be ordered before tx_2. We propose a protocol called `Garuda` that guarantees (γ,μ)-*approximate order-fairness* while incurring $\mathcal{O}(n)$ communication complexity without using any SNARKS in the critical path. This makes the protocol very practical and scalable.

As future work, we would need to incorporate sub-protocols for downward mobility and include upgrades that enable batch unspooling similar to that in Themis [103].

Bibliography

[1] libhotstuff: A general-purpose BFT state machine replication library with modularity and simplicity, 2018. https://github.com/hot-stuff/libhotstuff.

[2] Ethereum improvement proposal 1559, 2019. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md.

[3] Bitnodes network, 2020. Data drawn from website, https://bitnodes.earn.com/.

[4] Falcon, 2020. https://www.falcon-net.org/.

[5] Fibre, 2020. https://bitcoinfibre.org/.

[6] How many people use bitcoin in 2019?, 2020. https://www.bitcoinmarketjournal.com/how-many-people-use-bitcoin/.

[7] Compound finance, 2021. https://compound.finance/.

[8] Condorcet paradox, 2021. https://wikipedia.org/wiki/Condorcet_paradox.

[9] Curve, 2021. https://www.curve.fi/.

[10] Defi market capitalization, 2021. https://www.coingecko.com/en/defi.

[11] Defi pulse: The defi leaderboard, 2021.

[12] Maker, 2021. https://makerdao.com/.

[13] Smith criterion, 2021. https://en.wikipedia.org/wiki/Smith_criterion.

[14] Uniswap, 2021. https://uniswap.org/.

[15] BlockPI, Accessed 2023. https://blockpi.io/.

[16] DefiLlama Dashboard, Accessed 2023. defillama.com.

[17] Pocket network, Accessed 2023. https://www.pokt.network/.

[18] ZK System Benchmarking, Accessed 2023. https://github.com/delendum-xyz/zk-benchmarking.

[19] Osama Abboud, Aleksandra Kovacevic, Kalman Graffi, Konstantin Pussep, and Ralf Steinmetz. Underlay awareness in p2p systems: Techniques and challenges. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[20] Ittai Abraham, Yair Bartal, J Kleinberg, T-HH Chan, O Neiman, Kedar Dhamdhere, Aleksandrs Slivkins, and Anupam Gupta. Metric embeddings with relaxed guarantees. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 83–100. IEEE, 2005.

[21] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: simple and practical synchronous state machine replication. *IACR Cryptology ePrint Archive*, 2019:270, 2019.

[22] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder – scalable, robust anonymous committed broadcast. In *CCS*, page 1233–1252, 2020.

[23] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. Helix: A scalable and fair consensus algorithm resistant to ordering manipulation. Cryptology ePrint Archive, Report 2018/863, 2018. `https://eprint.iacr.org/2018/863`.

[24] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[25] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette. *arXiv preprint arXiv:1805.06786*, 2018.

[26] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930. ACM, 2018.

[27] Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof-of-stake longest chain protocols: Security vs predictability. *arXiv preprint arXiv:1910.02218*, 2019.

[28] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[29] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance, 2016. `https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf`.

[30] Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *ACM AFT*, pages 163–177, 2020.

[31] Rida Bazzi and Maurice Herlihy. Clairvoyant state machine replication, 2019.

[32] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.

[33] Lukas Bieri. Simulating bitcoin?s network topology. 2019.

[34] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, page 326–349, 2012.

[35] Blocknative. Evidence of mempool manipulation on black thursday: Hammerbots, mempool compression, and spontaneous stuck transactions, 2020.

[36] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[37] Lorenz Breidenbach, Phil Daian, Ari Juels, and Florian Tramèr. To sink frontrunners, send in the submarines, 2017. `https://hackingdistributed.com/2017/08/28/submarine-sends/`.

[38] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 459–473, 2019.

[39] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus, 2018.

[40] Benedikt Bünz and Ben Fisch. Transparent snarks from dark compilers. In *EUROCRYPT*, pages 677–706, 2020.

[41] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, page 524–541, 2001.

[42] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[43] Christian Cachin, Jovana Micic, and Nathalie Steinhauer. Quick order fairness. In *FC*, 2022.

[44] Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness, 2021.

[45] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 316–333. Springer, 2022.

[46] J-Y Cai, Richard J Lipton, Robert Sedgewick, and AC-C Yao. Towards uncheatable benchmarks. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11. IEEE, 1993.

[47] Paul Camion. Chemins et circuits hamiltoniens des graphes complets. *Comptes Rendus de l'Académie des Sciences de Paris*, 249:2151–2152, 1959.

[48] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–147, 2001.

[49] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[50] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[51] Chainlink. Smartcon 2022 research and product development highlights, 2022. `https://blog.chain.link/smartcon-research-updates/`.

[52] T-H. Hubert Chan, Naomi Ephraim, Antonio Marcedone, Andrew Morgan, Rafael Pass, and Elaine Shi. Blockchain with varying number of players. Cryptology ePrint Archive, Report 2020/677, 2020. `https://eprint.iacr.org/2020/677`.

[53] T-H Hubert Chan, Mingfei Li, Li Ning, and Shay Solomon. New doubling spanners: Better and simpler. *SIAM Journal on Computing*, 44(1):37–53, 2015.

[54] Nakul Chawla, Hans Walter Behrens, Darren Tapp, Dragan Boscovic, and K Selçuk Candan. Velocity: Scalability improvements in block propagation through rateless erasure coding. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 447–454. IEEE, 2019.

[55] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[56] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. *https://www.chia.net/assets/ChiaGreenPaper.pdf*, 2019.

[57] Coindesk. Ethermine adds front-running software to help miners offset eip 1559 revenue losses, 2021. `https://www.coindesk.com/ethermine-adds-front-running-software-to-help-miners-offset-eip-1559-revenu`

[58] Lin William Cong and Zhiguo He. Blockchain disruption and smart contracts. *The Review of Financial Studies*, 32(5):1754–1797, 2019.

[59] Thomas Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[60] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

[61] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004.

[62] Phil. Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *FC*, pages 23–41, 2019.

[63] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE S&P*, pages 585–602, 2020.

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

[65] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[66] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EuroCrypt*, pages 66–98, 2018.

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

[68] Anthony Dekker, Hebert Pérez-Rosés, Guillermo Pineda-Villavicencio, and Paul Watters. The maximum degree & diameter-bounded subgraph and its applications. *Journal of Mathematical Modelling and Algorithms*, 11(3):249–268, 2012.

[69] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. *ACM CCS, see also arXiv preprint arXiv:2005.10484*, 2020.

[70] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[71] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 21–30, 2011.

[72] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[73] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–154. Springer, 2020.

[74] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Trans- parent dishonesty: Front-running attacks on blockchain. In *FC*, pages 170–189, 2019.

[75] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In *FC*, pages 170–189, 2019.

[76] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Transparent dishonesty: front-running attacks on blockchain. In *3rd Workshop on Trusted Smart Contracts (WTSC)*, 2019.

[77] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, pages 436–454, 2014.

[78] Lei Fan and Hong-Sheng Zhou. A scalable proof-of-stake blockchain in the open setting (or, how to mimic nakamoto's design via proof-of-stake), 2018. Cryptology ePrint Archive, Report 2017/656, Version 20180425:201821.

[79] Guy Fayolle, Vadim Aleksandrovich Malyshev, and Mikhail Menshikov. *Topics in the constructive theory of countable Markov chains*. Cambridge university press, 1995.

[80] Ed Felton. Mev auctions considered harmful, 2020. `https://medium.com/offchainlabs/mev-auctions-considered-harmful-fa72f61a40ea`.

[81] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In *Theory of Cryptography Conference*, pages 322–352. Springer, 2020.

[82] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. Cryptology ePrint Archive, Report 2020/675, 2020. `https://ia.cr/2020/675`.

[83] Flashbots. Flashbots: Frontrunning the mev crisis, 2021. `https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752`.

[84] Nikolaos Fountoulakis, Anna Huber, and Konstantinos Panagiotou. Reliable broadcasting in random networks and the effect of density. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[85] Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and broadcast time of random geometric graphs in arbitrary dimensions. *Algorithmica*, 67(1):65–88, 2013.

[86] Alan Frieze and Wesley Pegden. Traveling in randomly embedded random graphs. *Random Structures & Algorithms*, 55(3):649–676, 2019.

[87] Juan Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In *CT-RSA*, pages 284–318, 2020.

[88] 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.

[89] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, pages 291–323, 2017.

[90] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. Full analysis of nakamoto consensus in bounded-delay networks. Cryptology ePrint Archive, Report 2020/277, 2020. `https://eprint.iacr.org/2020/277`.

[91] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[92] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. Cryptology ePrint Archive, Report 2016/1048, 2016. `https://eprint.iacr.org/2016/1048`.

[93] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705, 2015.

[94] 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 Principle*, pages 51–68, 2017.

[95] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[96] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[97] Chi Ho, Danny Dolev, and Robbert van Renesse. Making distributed systems robust. In *OPODIS*, pages 232–246, 2007.

[98] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. Measuring latency variation in the internet. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 473–480, 2016.

[99] Zi Hu, John Heidemann, and Yuri Pradkin. Towards geolocation of millions of ip addresses. In *Proceedings of the 2012 Internet Measurement Conference*, pages 123–130, 2012.

[100] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double spending fast payments on bitcoin. In *CCS*, pages 906–917, 2012.

[101] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE, 2000.

[102] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. `https://eprint.iacr.org/2021/139`.

[103] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

[104] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

[105] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

[106] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[107] 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.

[108] Ariah Klages-Mundt and Andreea Minca. (in)stability for the blockchain: Deleveraging spirals and stablecoin attacks, 2020.

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

[110] Klaus Kursawe. Wendy, the good little fairness widget. Cryptology ePrint Archive, Report 2020/885, 2020. `https://ia.cr/2020/885`.

[111] Klaus Kursawe. Wendy grows up, 2021. `https://vega.xyz/papers/Wendy_Grows_Up.pdf`.

[112] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *TOPLAS*, 4(3):382–401, 1982.

[113] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Fairledger: A fair blockchain protocol for financial institutions. In *OPODIS*, pages 4:1–4:17, 2019.

[114] Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. A decentralized blockchain with high throughput and fast confirmation. In *ATC*, pages 515–528, 2020.

[115] Jieyi Long and Ribao Wei. Nakamoto consensus with verifiable delay puzzle. *arXiv preprint arXiv:1908.06394*, 2019.

[116] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *CCS*, page 887–903, 2019.

[117] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005.

[118] Harsha V Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 367–380, 2006.

[119] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 373–388, 2013.

[120] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *CCS*, page 2111–2128, 2019.

[121] Y. Manoussakis. A linear-time algorithm for finding hamiltonian cycles in tournaments. *Discrete Appl. Math.*, 36(2):199–201, 1992.

[122] Alex Manuskin. The fastest draw on the blockchain: Ethereum backrunning, 2020. `https://medium.com/@amanusk/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1`.

[123] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. Cryptology ePrint Archive, Report 2018/581, 2018. `https://eprint.iacr.org/2018/581`.

[124] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[125] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin?s public topology and influential nodes. *et al*, 2015.

[126] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *CCS*, pages 31–42, 2016.

[127] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[128] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[129] Alex Obadia. Quantifying mev: Introducing mev-explore v0, 2021. https://medium.com/flashbots/quantifying-mev-introducing-mev-explore-v0-5ccbee0f6d02.

[130] A Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. Graphene: A new protocol for block propagation using set reconciliation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 420–428. Springer, 2017.

[131] R Pass, L Seeman, and A Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2017.

[132] Rafael Pass, Lior Seerman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, pages 643–673, 2017.

[133] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 315–324, 2017.

[134] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[135] Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *CSF*, pages 115–129, 2017.

[136] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

[137] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT*, pages 3–33, 2018.

[138] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[139] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest?, 2021.

[140] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE S&P*, pages 198–214, 2022.

[141] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[142] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In *FC*, 2021.

[143] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Trans. Program. Lang. Syst.*, 16(3):986–1009, 1994.

[144] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Trans. Program. Lang. Syst.*, 16(3):986–1009, 1994.

[145] Ling Ren. Analysis of nakamoto consensus. Technical report, Cryptology ePrint Archive, Report 2019/943.(2019). https://eprint. iacr. org . . . , 2019.

[146] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *TCC*, pages 262–285, 2016.

[147] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[148] Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 199–213, 2019.

[149] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[150] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[151] 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.

[152] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO*, pages 704–737, 2020.

[153] Zhan Shi. *Branching Random Walks*, volume 2151 of *Lecture Notes in Mathematics*. Springer Verlag, New York NY, 2015.

[154] Kiril Solovey, Oren Salzman, and Dan Halperin. New perspective on sampling-based motion planning via random geometric graphs. *The International Journal of Robotics Research*, 37(10):1117–1133, 2018.

[155] 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.

[156] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[157] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[158] Martino Trevisan, Danilo Giordano, Idilio Drago, Marco Mellia, and Maurizio Munafo. Five years at the edge: Watching internet from the isp network. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pages 1–12, 2018.

[159] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *IEEE S&P*, pages 926–943, 2018.

[160] X. et al Wang. Proof-of-stake longest chain protocol revisited. *arXiv preprint arXiv:1910.02218v2*, 2018.

[161] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2021.

[162] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, pages 1–35, 2020.

[163] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356, 2019.

[164] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.

[165] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. Ohie: blockchain scaling made simple. *arXiv preprint arXiv:1811.12628*, 2018.

[166] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *IEEE S&P*, pages 90–105, 2020.

[167] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *OSDI*, pages 633–649, 2020.

[168] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *OSDI*, pages 633–649, 2020.

[169] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE S&P*, 2021.

[170] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE S&P*, 2021.

[171] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445. IEEE, 2021.