DetTreeChain: Parallel Block Production on an N-ary Block Tree with Bounded Lane Divergence

A Preprint

Mahesh Karri* **and Nikhileswara Rao Sulake**†

Department of Computer Science and Engineering
Rajiv Gandhi University of Knowledge Technologies (RGUKT), Nuzvid, Andhra Pradesh, India

November 2026

Abstract

Nakamoto-style blockchains append one block at a time to a single chain, and raising the block rate increases forking and weakens security. We present DetTreeChain, a ledger in which up to N blocks are produced in parallel and placed without later merging or reordering. Blocks form an N-ary tree in which each block's slot follows from its parent and its lane. A locally derived Parent Queue Pool lets only two consecutive parents receive children, so lanes never drift more than one parent apart, and links between consecutive blocks of a lane make each lane a hash chain. As in several earlier designs, transactions are routed by what they spend, here a hash of the spent locking scripts, so blocks of different lanes cannot conflict and payments to other lanes need no cross-lane protocol. The rules are consensus-agnostic. Under proof of work, miners choose their lane, and an adversary with more than $1/(N+1)$ of the total hash power can outpace one lane. In a 25-minute run of our Rust prototype, six nodes converged on identical trees and difficulty targets, and 65% of slots were filled against 63% predicted. In simulation with delays of 5% of the lane interval, a single chain at the same block rate wasted 1.7 to 3.3 times as much work as 2 to 8 lanes, although nodes more often held later-discarded blocks. Among Bitcoin transactions whose spent scripts could be rebuilt, about 95% kept their inputs in one lane, mostly because they spend a single output.

Keywords blockchain scalability · parallel block production · proof of work · transaction routing · UTXO · distributed ledger

1 Introduction

Bitcoin orders transactions by appending blocks to a single chain, one block at a time [1]. The design has run reliably since 2009, but its throughput depends on two parameters, the block size and the block interval. Both are limited by the time a block takes to reach the rest of the network [2]. If blocks come faster, or grow larger, than the network can spread them, concurrent blocks fork the chain. Honest work is wasted, and an attacker needs a smaller share of the hash power [3, 4].

Formal analyses of the longest-chain rule agree: its security degrades as the block rate grows relative to the network delay [5–7]. As Bagaria et al. argue, a Nakamoto chain therefore runs out of security long before it runs out of bandwidth [8]. A natural remedy is to let several blocks be produced at the same time. Block DAGs keep concurrent blocks and order them later [9, 10]. Parallel-chain protocols run several chains side by side and merge them into one log [8, 11, 12]. Sharded ledgers split nodes, and often state, into groups that work in parallel [13–16].

Each of these approaches must answer two questions that a single chain never faces. The first, a question of structure, is where a concurrently produced block belongs and how far apart its producers may drift. The second, a question of content, is whether two concurrent blocks may contain conflicting transactions and who resolves the conflict if they do.

*Corresponding author: maheshchinna6306@gmail.com
†nikhil01446@gmail.com

Most designs answer at least one of these questions after the blocks exist, by ordering a DAG, by merging chains, or by running a commit protocol across shards.

In this paper we aim to settle both questions before a block is produced, using rules that every node checks locally. We present DetTreeChain, a ledger organised as an N-ary tree of blocks (Figure 1). Every parent block offers N child slots, one per lane. The slot of a block follows from its parent and its lane, so all nodes place a given block at the same position. A Parent Queue Pool (PQP), which each node derives from the blocks it holds, opens a window of only two consecutive parents for children. When the newer parent receives a child, the older one closes, and no lane can run more than one parent ahead of the others. To the best of our knowledge, no earlier design places concurrent blocks by the slot numbering of a single tree and bounds the drift between producers with a window that each node derives locally.

A block becomes a parent long after it is created, so tree edges alone leave recent blocks unprotected. Each block therefore also links to the previous block of its own lane, which turns every lane into a hash chain. Transactions are routed to lanes by the locking scripts of the outputs they spend, so two transactions that spend the same output always meet in one lane. Partitioning transactions by what they spend is an established idea. Blockclique and Zilliqa route a transaction by its sender's address, Monoxide partitions accounts by address, and Eunomia tags each output with the chain where it may be spent [16–19]. We derive the lane of an unspent transaction output (UTXO) from its locking script, so no index has to be chosen or stored.

DetTreeChain is an architecture for the ledger, not a new consensus protocol. It fixes where blocks go and which parents may receive children, and leaves the choice of producer to a consensus layer. The "determinism" in its name refers to the placement of blocks.

We built the proof-of-work instantiation first because it is, in our view, the hardest case for the structure: nobody is scheduled, block times are random, and lanes race for slots. It also exposes the main cost of the design. When miners choose their lane, an adversary can concentrate on one lane, and the share of total hash power it needs to outpace that lane falls from $1/2$ to $1/(N+1)$. This is the familiar dilution of hash power across parallel chains and shards [11, 16, 20], and we quantify it. Every node still receives and validates every block, so, as in any fully replicated ledger, end-to-end throughput stays bounded by one node's bandwidth and validation capacity [21, 22].

This paper makes four contributions.

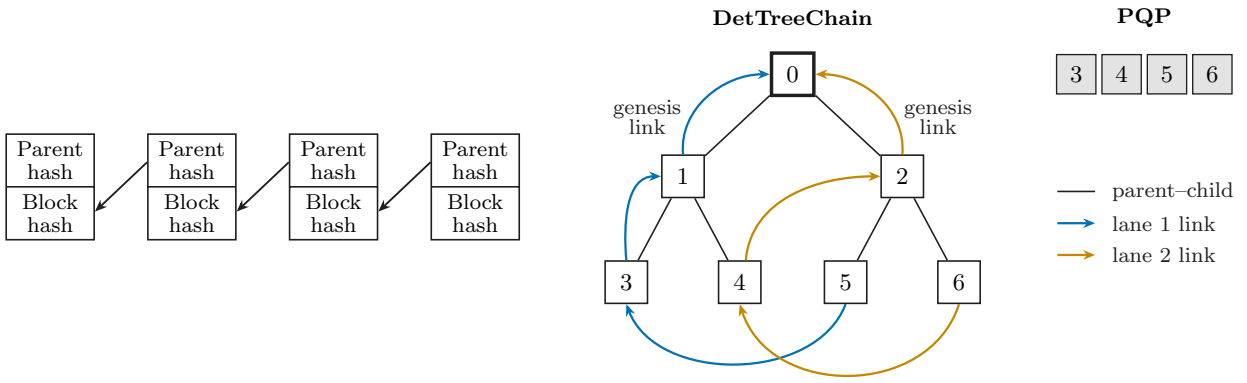
- **Design.** We specify slot numbering, the PQP and its two-parent window, lane links, and an interface to the consensus layer, together with the fork-choice, difficulty, and reward rules for proof of work (Section 4).
- **Conflict-free routing.** We route UTXO transactions by a hash of the locking scripts they spend. We show that blocks of different lanes cannot contain conflicting transactions, and that payments to other lanes need no cross-lane protocol when every node stores the full UTXO set (Section 5).
- **Analysis.** We show that lanes stay within one parent of each other and that every lane is a hash chain, under any consensus layer. For proof of work, we derive the expected fill rate of the tree in closed form and quantify the cost of miner-chosen lanes (Section 6).
- **Prototype and evaluation.** We describe a Rust prototype and evaluate it with a six-node testbed, a discrete-event simulation under propagation delay, and a routing study of 762,842 Bitcoin transactions (Sections 7 and 8). We relate the results to measured delays and stale rates of Bitcoin and Ethereum (Section 9).

The evaluation asks four questions. *RQ1:* Do independent nodes build the same tree and compute the same difficulty? *RQ2:* How full does the tree become, and does our model predict it? *RQ3:* How much mined work does DetTreeChain waste under propagation delay, compared with a single chain at the same block rate? *RQ4:* How do real owners and their transactions spread over lanes, and how many of today's transactions already satisfy the one-lane rule? In our experiments, all six nodes ended with identical trees and difficulty targets, and the fill rate lay about one standard deviation above the model. In simulation, with delays of 5% of the lane interval, a single chain at the same block rate wasted 1.7 to 3.3 times as much work as DetTreeChain with 2 to 8 lanes. About 95% of the Bitcoin transactions we could analyse already kept their inputs in one lane.

2 Background and Related Work

2.1 Nakamoto consensus and its limits

In Nakamoto consensus, miners extend the chain with the most accumulated work, and a transaction is settled once enough blocks bury it [1]. Garay et al. showed that such a chain has common prefix, chain quality, and chain growth while honest miners hold a majority and blocks are rare compared with the network delay [5]. Later work



(a) A linear chain

(b) DetTreeChain with $N = 2$

Figure 1: (a) In a linear chain, each block references one predecessor. (b) In DetTreeChain with $N = 2$, straight edges connect parents and children, and curved arrows are lane links from each block to the previous block of its lane. The first block of each lane links to the genesis block. The PQP lists the blocks that can still receive children.

covered networks with bounded delay and tightened the security threshold as a function of block rate and delay [6, 7]. Measurements agree that larger or more frequent blocks raise the stale rate and make double spending cheaper [3, 4].

Bitcoin-NG separates leader election from the serialisation of transactions [23]. Neu et al. showed that longest-chain protocols need care in how blocks are downloaded once bandwidth becomes the bottleneck [21]. All of these designs keep a single chain at the core.

2.2 Block DAGs and block trees

GHOST keeps forked blocks in a tree and picks the main chain by subtree weight [24]. Block DAGs go further: every block references all the tips its producer knows [25]. GHOSTDAG orders such a DAG greedily [9], and Conflux adds reference edges to a pivot chain so that the transactions of concurrent blocks are all processed [10]. Because any miner may extend any tip of a proof-of-work DAG, the shape of the graph depends on network timing, and conflicts between concurrent blocks are resolved only after ordering.

Trees play other roles too. Tailstorm arranges the proofs of work of one block in a tree to speed up confirmation and discourage withholding [26]. Tree-chain, built for Internet-of-Things deployments, gives each validator a branch and routes each transaction by a prefix of its hash [27, 28]. Because the route depends on the transaction and not on the outputs it spends, the transactions of one owner, even two that spend the same output, can land on different branches, and the branches share no slot numbering or window. DetTreeChain instead routes by the owner of the spent outputs, so a double spend always meets in one lane. Unlike the tree of forks in GHOST, the tree of DetTreeChain is the ledger itself, and its slots are the unit of parallelism.

2.3 Parallel chains

OHIE runs several Bitcoin-style chains and assigns each block to a chain by its hash, so a miner learns its chain only after solving the puzzle [11]. Confirmed blocks of all chains are then merged into one sequence. Prism confirms proposer blocks through many voter chains [8], and Fitzi et al. combine several ledgers into one with faster settlement [12]. These protocols keep the security threshold of a single chain while raising throughput.

Other designs coordinate the chains more tightly. In Chainweb, a block at height h references the blocks at height $h - 1$ on neighbouring chains. This braiding bounds how far a chain can run ahead, and an attacker must rewrite the neighbours too [29, 30]. Eunomia combines OHIE-style assignment with UTXO sharding. Each UTXO carries the index of the chain where it may be spent, chosen by the payer, and a spend of a coin from another chain is confirmed once the block that created the coin is confirmed [19]. Blockclique arranges blocks in threads with clock-driven slots and assigned producers. Every block references one block of each thread, and may include only transactions whose paying address belongs to its thread [17].

Parallel chains also complicate difficulty adjustment, which Garay et al. analysed for a single chain [31]. Wang et al. showed that natural adaptations of Bitcoin's rule to parallel chains can break safety or liveness when hash power changes, and proposed a rule based on a pivot chain [32].

2.4 Sharding and transaction placement

Sharding divides the work of a ledger among groups of nodes. One taxonomy separates network, transaction, and state sharding [33]. Elastico assigns nodes to committees that process disjoint sets of transactions, which a final committee merges [13]. OmniLedger and RapidChain also shard the state, so transactions that touch several shards need a cross-shard commit protocol [14, 15]. Monoxide partitions accounts by address into zones that each run a proof-of-work chain. A transfer between zones completes through a relay transaction, and Chu-ko-nu mining makes a single zone as hard to attack as the whole network [16].

Zilliqa assigns transactions to committee-run shards by bits of the sender's address, so double spends meet in one shard [18]. Later systems lower the cost of cross-shard transactions or make small shards safe [34, 35]. When nodes choose their own shard, an adversary can concentrate on one, and Free2Shard studies allocation rules that resist such concentration [20].

A related line of work asks where transactions should go. OptChain places Bitcoin transactions to reduce cross-shard dependencies, and reports that random placement over 16 shards made 93.5% of 10 million Bitcoin transactions cross-shard [36]. TxAllo allocates Ethereum accounts using the transaction graph [37], and LB-Chain shows that uneven load across shards drives confirmation delay more than the cross-shard ratio does [38]. In UTXO state sharding, a transaction is cross-shard when its inputs or its outputs lie in different shards, because each shard stores only its own outputs.

2.5 Owner-partitioned payments and committee protocols

Guerraoui et al. showed that transferring assets with a single owner does not need consensus [39], and FastPay built a payment system on this result [40]. Sui separates owned objects, sequenced by their owner, from shared objects, ordered by consensus [41]. Parallel execution engines target a different bottleneck: they run the transactions of an already ordered block concurrently [42].

Committee-based protocols build a DAG of validator blocks and order it with Byzantine fault-tolerant consensus [43, 44]. Autobahn disseminates data in per-validator lanes whose tips a separate consensus layer orders [45]. These protocols reach high throughput and deterministic finality, but assume a known committee with fewer than a third faulty members. An Autobahn lane carries one validator's data, and conflicts across lanes are resolved after ordering. A DetTreeChain lane is instead defined by the owners of the outputs it may spend.

2.6 Positioning

DetTreeChain builds on three existing ideas (Table 1): fixed positions for concurrent blocks, a bound on how far producers drift apart, and routing by owner so that parallel blocks cannot conflict. Its contribution is the way it combines them. Positions come from the slot numbering of one tree, with no clock, producer schedule, or committee. The drift between lanes is bounded by a window that each node derives locally, without references between lanes. Routing works on UTXOs, through the locking scripts of the spent outputs. By comparison, Blockclique fixes positions with a clock and assigned producers, Eunomia assigns blocks by hash and orders them afterwards, and Chainweb bounds drift through references to neighbouring chains.

DetTreeChain is not state sharding. Every node stores and validates all lanes, so the design partitions transactions and block production but not storage. Full replication is what lets payments to other lanes proceed without a cross-lane protocol, and why we call the partitions lanes instead of shards. Under proof of work, the higher block rate costs security per lane (Section 6.4), a cost that OHIE, Prism, and Eunomia avoid by assigning blocks to chains by hash.

3 System Model and Goals

Participants and network. Nodes form a peer-to-peer network, and every node stores and validates the whole ledger. Some nodes also produce blocks, and users spend outputs with transactions. We adopt the bounded-delay model [6]: a message from an honest node reaches every other honest node within Δ seconds, and the adversary may delay it up to that bound.

Consensus layer. DetTreeChain fixes the structure of the ledger and leaves the choice of producers to a consensus layer (Section 4.7). Structural results hold for any consensus layer, and we mark results that depend on proof of work. In the proof-of-work instantiation, all lanes share one difficulty target T, and blocks are found as a Poisson process of total rate λ. We assume that honest miners spread their hash power evenly over the N lanes, so honest blocks of

Table 1: Comparison of DetTreeChain with related designs.

System	Ledger structure	Position of a concurrent block	Transaction partitioning	Payment to another partition	Security of one partition
Bitcoin [1]	single chain	no concurrent blocks	none	not applicable	all hash power
GHOSTDAG [9], Conflux [10]	block DAG	decided by ordering the DAG	none; conflicts settled after ordering	not applicable	all hash power
OHIE [11]	parallel chains	chain chosen by block hash	none	not applicable	adversary's global share on every chain
Chainweb [29]	braided parallel chains	chain and height	per chain, chosen by the user	cross-chain transfer with proofs	braiding with neighbouring chains
Eunomia [19]	parallel chains	chain chosen by block hash	UTXO carries a chain index chosen by the payer	spent at once; confirmed after the creating block	adversary's global share on every chain
Blockclique [17]	threads; each block references every thread	thread and clock slot	by paying address	none needed	producers assigned to slots
Monoxide [16]	independent zones	zone	by account address	relay transaction	Chu-ko-nu mining
Zilliqa [18]	shard microblocks merged by a committee	shard and epoch	by sender address	handled in the sender's shard	committee per shard
Tree-chain [27, 28]	one branch per validator, renewed each epoch	branch and epoch	by prefix of the transaction hash	not applicable	one validator per branch, no mining
Narwhal and Bullshark [43, 44]	DAG of validator blocks	validator and round	none; conflicts settled after ordering	not applicable	committee, fewer than $n/3$ faulty
DetTreeChain (this work)	N-ary tree with lane chains	slot fixed by parent and lane	by hash of the spent outputs' locking scripts	none needed (full replication)	set by the consensus layer; under proof of work, the hash power of the lane

one lane arrive at rate $(1-\beta)\lambda/N$, where β is the adversary's share. Every block a miner finds has a valid slot, so the subsidy alone favours no allocation. Fees follow the backlog of each lane and may pull hash power towards congested lanes (Section 9).

Adversary. The adversary controls a fraction $\beta < 1/2$ of the total hash power. It may concentrate this power on any lane, withhold blocks, and delay honest messages within Δ. It cannot find SHA-256 collisions or forge signatures. Our analysis uses informal arguments and proof sketches, and a treatment in the frameworks of Garay et al. and Pass et al. is future work [5, 6].

Goals. The design pursues four goals.

G1 *Local placement.* Every node computes the position of a block from the block and the blocks it already holds, without a clock, a producer schedule, or extra messages.

G2 *Bounded divergence.* At any time, the producers of all lanes work on children of at most two consecutive parents.

G3 *Conflict freedom.* Blocks of different lanes never contain two transactions that spend the same output, and paying an owner in another lane needs no cross-lane protocol.

G4 *Independence from the consensus layer.* Every node checks the structure in the same way under any consensus mechanism that supplies a producer proof and a fork-choice rule.

DetTreeChain does not try to reduce the storage, bandwidth, or validation work of a node. Table 2 lists our notation.

Table 2: Notation.

Symbol	Meaning
N	number of lanes, equal to the branching factor of the tree
g	genesis block
$a(b)$, $\mathrm{par}(b)$, $s(b)$	lane, parent, and slot index of block b
$\mathrm{pred}(b)$	lane predecessor of b: the accepted block of the same lane with the largest smaller slot
$H(b)$, $C(b)$, $\ell(b)$, $\sigma(b)$	block hash, lane commitment, lane link, and producer signature of b
c, c'	current parent and next parent, the first two entries of the PQP
$script(o)$, $\mathrm{lane}(o)$, h	locking script of output o, the lane it is routed to, and the 64-bit script hash
β, β_L	adversary's share of the total hash power, and its share of the hash power on an attacked lane
λ, Δ	total block rate and bound on message delay between honest nodes
T, I, R	difficulty target, target interval between blocks of one lane, and blocks per lane in a retarget period
k, ε	confirmation depth counted in blocks of one lane, and target failure probability
ϕ	fill rate: the fraction of slots under retired parents that hold a block

4 DetTreeChain Design

4.1 Overview

DetTreeChain replaces the chain of a UTXO blockchain with a rooted tree whose branching factor $N \geq 1$ is a protocol constant. With $N = 1$ the tree is an ordinary chain. Every block b except the genesis block g has a parent $\mathrm{par}(b)$ and a lane $a(b) \in \{1, \ldots, N\}$, and b is the $a(b)$-th child of its parent. A producer works for one lane only. Three mechanisms coordinate the lanes: slot numbering places every block (Section 4.2), lane links connect consecutive blocks of a lane (Section 4.3), and the PQP decides which parents may receive children (Sections 4.4 and 4.5). The consensus layer decides who may fill a slot and which of two competing trees to keep (Sections 4.6 and 4.7).

4.2 Slots

Definition 1 (Slot index)**.** The genesis block has slot $s(g) = 0$. Every other block b has slot

$$s(b) = N \cdot s(\mathrm{par}(b)) + a(b). \tag{1}$$

Equation (1) is the breadth-first numbering of a complete N-ary tree. The children of the block in slot p occupy slots $Np + 1$ to $Np + N$. The slot of a block thus depends only on its parent and lane, and two blocks with the same parent and lane compete for one slot. Since $s(b) \equiv a(b) \pmod N$, the lane of a block can be read from its slot. Each block also carries its depth and the sequence of lanes on its path from the root, and validators check that these agree with its slot. Slots that never receive a block stay empty and are a normal part of the structure, as Section 6.3 shows.

4.3 Block contents and lane links

Besides the usual header fields and its producer proof (under proof of work, a compact target and a nonce), a block holds its parent hash, lane, depth, path, and a PQP entry. The PQP entry contains the slot $s(b)$, the producer's address, a lane link $\ell(b)$, and the producer's signature $\sigma(b)$ over these fields and the parent hash. The block hash $H(b)$ is a SHA-256 digest of the header, the PQP entry, and the transaction identifiers. Under proof of work, a valid block also satisfies $H(b) < T$. The lane commitment of a block is

$$C(b) = \mathrm{SHA256}\big(s(b) \,\|\, a(b) \,\|\, H(b) \,\|\, \mathrm{par}(b) \,\|\, \mathit{producer} \,\|\, \ell(b) \,\|\, \sigma(b)\big). \tag{2}$$

Definition 2 (Lane predecessor)**.** The lane predecessor $\mathrm{pred}(b)$ of a block b is the accepted block b' with the largest slot $s(b') < s(b)$ such that $s(b') \equiv s(b) \pmod N$. If no such block exists, $\mathrm{pred}(b) = g$.

A valid block links to the commitment of its lane predecessor:

$$\ell(b) = C(\mathrm{pred}(b)). \tag{3}$$

The parent hash fixes the place of a block in the tree, and the lane link ties it to the previous block of its lane. The first block of each lane links to the genesis block (Figure 1b), and a lane that misses a slot links its next block to the last block it did produce. Lane chains therefore have no gaps, even where the tree has them.

The second link exists because parent references point far into the past. In a full tree, the parent of a new block was created when the tree held about $1/N$ of its current blocks, and the new block becomes a parent only after the tree has grown about N-fold (Section 6.2). Without lane links, nothing would build on a new block for a long time. With them, each new block extends its lane at once, and every lane behaves like a chain for confirmation.

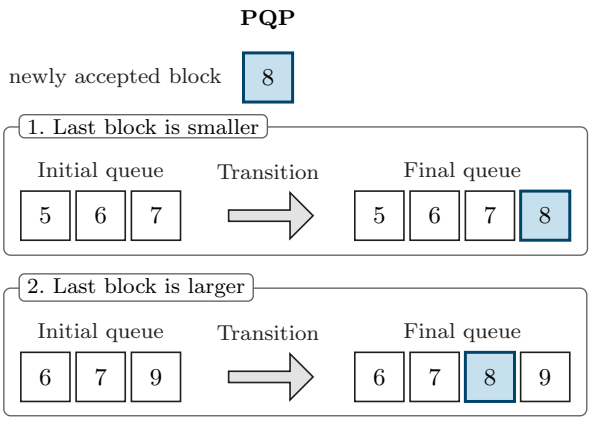


Figure 2: Insertion into the PQP. A newly accepted block (8) is appended when its slot is larger than that of the last entry (top) and inserted at its sorted position otherwise (bottom). The first two entries are the current and the next parent.

4.4 Parent Queue Pool and acceptance rules

The PQP is an ordered list of accepted blocks that may still receive children. Every node maintains it locally and can rebuild it from the blocks it holds, so it is never sent over the network. Entries are ordered by slot (Figure 2), and the first two are the current parent c and the next parent c'. Two rules govern which blocks a node accepts.

Rule 1 (Window)**.** A node accepts a block b only if $\mathrm{par}(b) \in \{c, c'\}$ in its current PQP, the slot $s(b)$ is empty, and the lane link of b satisfies Eq. (3).

Rule 2 (Retirement)**.** When c has N children, it leaves the PQP. When c' receives its first child, c leaves the PQP even if some of its slots are still empty, and c' becomes the current parent.

Every accepted block enters the PQP as a future parent. We call $\{c, c'\}$ the *window*, the children of one parent a *sibling group*, and the period during which a block is the current parent a *parent cycle*. A sibling group closes when its parent leaves the PQP. Algorithm 1 lists the checks a node applies to an incoming block. Section 5 describes the checks on its transactions.

Each node applies the two rules in the order in which blocks arrive. A node that has already seen the first child of c' rejects a later block for c, while a node that received that block earlier keeps it. A node that joins later cannot know either order. It therefore checks a complete tree by replaying its blocks in slot order, in which every child of c precedes the children of c'. A late block then passes unless a later block of its own lane links past it. Since no tree can contain both blocks, the two versions form a fork, which the fork-choice rule settles as Bitcoin settles an orphaned block (Section 4.6).

4.5 The Parent-Child Completion Constraint

We call Rule 2 the Parent-Child Completion Constraint. A producer of lane a first tries to fill slot $Ns(c) + a$. Once its lane holds a child of c, it moves to slot $Ns(c') + a$ and links to that child. Producers of slower lanes keep working on c, but only until some lane places a child under c' and c leaves the PQP (Figure 3). A block for c that arrives later names a parent outside the window, and its lane continues from its last accepted block.

The constraint bounds divergence without any extra message: all lanes always work on children of at most two consecutive parents (Proposition 2). Its cost is that slow lanes can lose slots, and under proof of work this happens even without delay. A lane that has already filled its slot under c may find its next block, under c', before a slower lane finds its first, and that block closes c. Section 6.3 computes how many slots stay empty. Since proof of work is memoryless, an empty slot wastes no hashing work: a miner whose parent retires simply moves to the next one.

4.6 Fork choice and finality

Since the rules are applied in arrival order, different nodes can accept different blocks. This happens when two producers of one lane fill the same slot at almost the same time, or when a late block for c and the first child of c' reach nodes in opposite orders. Within a node, the first valid block for a slot wins, as in Bitcoin. Across nodes, the consensus layer decides.

Algorithm 1 Handling of an incoming block b at a node

Require: local tree; PQP with current parent c and next parent c'
if the producer proof, $\sigma(b)$, or $C(b)$ is invalid **then** ▷ proof of work: $H(b) < T$
  return reject
end if
$p \leftarrow \mathrm{par}(b)$
if $p \notin \{c, c'\}$ **then** ▷ Rule 1
  return reject
end if
if $s(b) \neq N\, s(p) + a(b)$ **or** slot $s(b)$ is occupied **then**
  return reject
end if
if $\ell(b) \neq C(\mathrm{pred}(b))$ **then** ▷ lane link
  return reject
end if
if a transaction is invalid or belongs to another lane **then** ▷ Section 5
  return reject
end if
store b at slot $s(b)$ and insert it into the PQP by slot
if $p = c'$ **or** c now has N children **then** ▷ Rule 2
  remove c from the PQP
end if
return accept

Our proof-of-work instantiation uses a longest-tree rule. A node that learns of a valid tree with more blocks than its own adopts it. Between trees of equal size, it prefers the one whose hash over its block hashes, taken in slot order, is smaller, so all nodes break ties alike. Within one retarget period all blocks share a target, so block count tracks expected work, as in Nakamoto's heaviest-chain rule. Across periods this no longer holds, and comparing accumulated work, as in the analysis of chains with variable difficulty, is safer [31], although our prototype still counts blocks. Finality is probabilistic: a transaction becomes harder to reverse as blocks of its lane accumulate after it (Section 6.4).

The determinism in DetTreeChain concerns structure, not finality. Given a set of blocks, the slots, the lane chains, and the PQP are uniquely determined. Two nodes that have accepted the same blocks therefore hold the same tree and PQP, whatever the arrival order. Which blocks a node accepts can still depend on that order (Section 4.4).

4.7 Interface to the consensus layer

The structure consists of the slots, the lane links, the PQP, and the routing rule of Section 5. Every node checks it in the same way under any consensus mechanism (goal G4). The consensus layer supplies a producer proof, a fork-choice rule for competing valid trees, and optionally a finality rule. Three instantiations are natural.

- *Proof of work.* Miners choose a lane and race for its free slots. The producer proof is a block hash below the target, the longest-tree rule resolves forks, and finality is probabilistic. This is the instantiation we built.
- *Proof of stake with slot leaders.* A leader is drawn for every slot from the stake distribution, for example with verifiable random functions as in Algorand [46]. A slot then stays empty only if its leader is late or offline, and the window acts as a timeout. If all lanes draw leaders from the total stake, an adversary leads each lane roughly in proportion to its stake and cannot pick one lane to attack.
- *Committee-based consensus.* A validator committee certifies the children of each parent with a quorum certificate [47], which would give deterministic finality for every parent cycle.

Hybrid designs are possible too, such as proof of work for producing blocks and a committee for finalising parent cycles. Apart from proof of work, these options are sketches that show what the interface must provide; we have not implemented them.

4.8 Difficulty and rewards under proof of work

All lanes share one difficulty target. After every NR accepted blocks, the target is rescaled by the ratio between the time those blocks took and the expected time RI. As in Bitcoin, the ratio is clamped to $[1/4, 4]$, a dampening filter

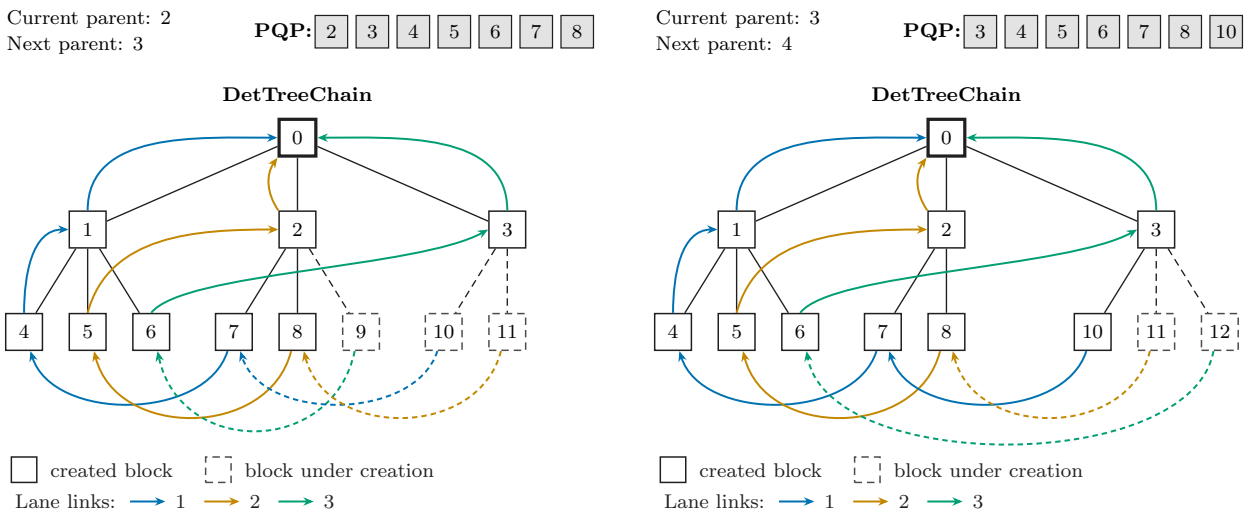


Figure 3: The Parent-Child Completion Constraint for $N = 3$. The PQP (top) lists the blocks that can still receive children, and dashed boxes are blocks still being produced. (a) Lanes 1 and 2 have placed blocks 7 and 8 under the current parent 2 and are working on slots 10 and 11 under the next parent 3, while lane 3 is still working on slot 9. (b) Block 10, a child of parent 3, is accepted first. Parent 2 leaves the PQP and slot 9 stays empty. Lane 3 now works on slot 12 and links to block 6, the last block of its lane.

in the terms of Garay et al. [31]. The tree thus aims at N blocks per interval I, one per lane. Because the rule counts blocks instead of slots, empty slots do not distort the schedule. The subsidy likewise halves after a fixed number of blocks, and coinbase outputs become spendable after a maturity delay.

Neither count may depend on the order in which blocks arrive. The count of a block is therefore the number of accepted blocks before its sibling group plus its lane index. The target of a period is computed from the previous target and from the timestamps of blocks that precede every sibling group of that period. All nodes that hold those blocks compute the same target, whatever the order of later blocks. Wang et al. note that aggregating inter-block times across parallel chains needs some synchronisation between the chains [32]. In DetTreeChain the closed sibling groups of the PQP can provide it. The security of this rule under changing hash power, and the validation of timestamps, remain open (Section 6.6). Stake-based and committee-based instantiations would replace the target with their leader schedule.

5 Transactions

5.1 Owners and routing

DetTreeChain uses Bitcoin's UTXO model, in which each output is locked by a script that states who may spend it. We treat the locking script $script(o)$ of an output o as its owner. All outputs sent to one address carry the same locking script, whether the address belongs to a single key, a multisignature group, or a timelocked script. Routing assigns each output to the lane

$$\text{lane}(o) = \big(h(script(o)) \bmod N\big) + 1, \tag{4}$$

where h takes the first 64 bits of the SHA-256 hash of the script. Every node can compute the lane of an output as soon as the output exists, without any coordination.

Rule 3 (Routing)**.** A transaction is valid only if all its inputs belong to the same lane a, and it may be included only in blocks b with $a(b) = a$.

In this paper, a *conflict* means two transactions that spend the same output, that is, a double spend. Parallel execution uses a broader notion, in which transactions conflict whenever their read and write sets overlap [42]. A *cross-lane transaction* has inputs in different lanes, and Rule 3 forbids it. A *cross-lane payment* creates outputs in other lanes, and it is allowed.

5.2 Conflict freedom

Proposition 1 (Conflict freedom)**.** *Under Rule 3, any two transactions that spend a common output belong to the same lane.*

Proof. Let tx_1 and tx_2 both spend an output o. By Rule 3, each transaction belongs to the lane of each of its inputs, and in particular to $\mathrm{lane}(o)$. By Eq. (4), $\mathrm{lane}(o)$ depends only on $script(o)$, which is part of the output and hence the same for every transaction that spends it. Both transactions therefore belong to $\mathrm{lane}(o)$. □

Corollary 1. *Blocks of different lanes, and in particular children of the same parent, never contain conflicting transactions. Conflicts can arise only within one lane, which its lane chain orders, so ordinary UTXO validation rejects the later spend.*

The proof is short because the property is built into the routing rule. It reflects a general result: transfers of assets with a single owner need only that owner's sequencing and no global agreement [39].

5.3 Payments across lanes

Suppose Alice, whose outputs lie in lane 1, pays Bob, whose addresses map to lane 3. The payment is an ordinary lane-1 transaction that consumes Alice's outputs and creates an output locked to one of Bob's addresses, which every node recognises as a lane-3 output. Nothing is written in lane 3. When Bob later spends the output, Rule 3 routes his transaction to lane 3, and lane-3 producers already hold the output. No relay transaction or atomic commit is needed. Monoxide, in contrast, debits one zone and credits another [16], Chainweb moves assets with cross-chain proofs [29], and Eunomia confirms a spend of a coin from another chain only after the block that created it [19]. The price is full replication: every node stores and validates the entire tree (Section 9).

Outputs created by concurrent blocks need one refinement. Suppose a lane-3 block spends an output that a lane-1 sibling under the same parent created. Its validity at a node then depends on whether the node has already received the sibling. DetTreeChain therefore lets another lane spend an output only after the sibling group that created it has closed. Our prototype does not enforce this rule yet (Section 7). Spends within a lane need no delay, because the lane link already orders them. A stronger version of the rule also limits how far reorganisations spread across lanes (Section 6.5).

5.4 Lane-aware wallets

A wallet can keep all its coins in one lane, its *home lane*, by handing out only addresses that map to it. Since h behaves like a random function, finding such an address takes N key generations on average. The wallet can then combine any of its coins, including change, in one transaction. Hashing the whole script also spreads owners evenly over the lanes for every N, which a rule based on one digit of the public key does not (Section 8.5).

Home lanes have costs. All addresses of a wallet reveal the same $\log_2 N$ bits, which adds to existing heuristics that link the addresses of one owner, such as the assumption that all inputs of a transaction belong to one entity [48]. Hierarchical wallets must also derive about N times as many keys and raise their gap limit accordingly. Transactions that combine inputs of several parties, such as CoinJoin, need all parties to share a lane or must be split. Section 8.5 measures how many of today's Bitcoin transactions already satisfy Rule 3.

5.5 Beyond UTXO

Account-based state fits lanes less naturally. A payment debits the sender and credits the receiver in place, so a credit written in the sender's lane can race with a debit in the receiver's lane. Two remedies seem compatible with the design. Credits to accounts of other lanes could be applied at the end of the parent cycle, since additions commute and debits stay in the owner's lane. Alternatively, assets could be modelled as objects with a single owner, as in Sui [41]. The new owner then uses the object in its own lane, and we expect Proposition 1 to carry over. Contracts whose state many users share would need a lane of their own or execution deferred to the end of each parent cycle. The prototype implements neither.

6 Analysis

Sections 6.1 and 6.2, like Proposition 1, rely only on the slots, the PQP, and the routing rule, and hold under any consensus layer. Sections 6.3 to 6.6 concern proof of work, where producers choose their lane and block times are random.

6.1 Structural properties

Proposition 2 (Slot uniqueness and bounded divergence)**.** *At every honest node, (a) each slot holds at most one block, (b) each parent has at most one child per lane, and (c) every block accepted while c is the current parent is a child of c or of the next parent c'. Consequently, the lanes always produce children of at most two consecutive parents.*

Proof sketch. Parts (a) and (b) follow from Eq. (1) and Rule 1, because a second block with the same parent and lane would target an occupied slot. For (c), Rule 1 admits only children of c and c'. When c' receives a child, Rule 2 retires c, so the window moves forward by exactly one parent. A lane that already has a child of c' can continue only under the parent after c', and only once c' has become the current parent. □

Proposition 3 (Lane chains)**.** *For every lane, the accepted blocks of the lane, ordered by slot, form a hash chain. Replacing a block of a lane invalidates the lane links of all later blocks of that lane.*

Proof sketch. Each block commits to its lane predecessor through $\ell(b) = C(\mathrm{pred}(b))$ (Definition 2 and Eq. (3)). By Eq. (2), $C(b)$ depends on $H(b)$, which covers $\ell(b)$. A change to any block therefore changes the commitment that its lane successor must link to. □

6.2 Age of parent references

Observation 1. *Suppose every slot is filled, and number the parent cycles from 1. A block in slot p is created during cycle $\lceil p/N \rceil$ and becomes the current parent in cycle $p + 1$. A block created in cycle t is therefore first used as a parent around cycle Nt, and its own parent was created around cycle t/N.*

Empty slots change the constants but leave the conclusion intact. If a fraction ϕ of the slots is filled, the effective branching factor is ϕN, and the parent of a block created in cycle t dates from around cycle $t/(\phi N)$. Between cycle t and roughly cycle $\phi N t$, only later blocks of its own lane depend on a block created in cycle t. Tree links thus add no protection to recent blocks, and confirmation depth must be counted in lane blocks. This is why lane links exist, and why, under proof of work, the security of a transaction depends on the hash power of its lane.

6.3 Empty slots under proof of work

Consider proof of work with equal hash power in every lane and no delay, in which each new block comes from a uniformly random lane. A lane without a child of c produces one. A lane that already has one produces the first child of c', which closes c. Parent c thus collects children until some lane is drawn a second time, exactly as in the birthday problem. Every parent starts alike: its first child comes from the draw that closed the previous parent, or from the next draw if that parent filled all N slots. Since draws are independent and lanes symmetric, the numbers of children of successive parents are independent and identically distributed.

Proposition 4 (Expected fill rate)**.** *With N lanes of equal hash power and no network delay, the expected number of children of a parent is*

$$Q(N) = \sum_{j=1}^{N} \frac{N!}{(N-j)!\, N^j}, \tag{5}$$

so the expected fill rate is $\phi = Q(N)/N$, and a parent receives a single child with probability $1/N$.

Proof sketch. A parent receives at least j children exactly when the first j draws hit distinct lanes, which happens with probability $N!/((N-j)!\, N^j)$. Summing over $j = 1, \ldots, N$ gives the expectation. A parent receives one child exactly when the second draw repeats the first, with probability $1/N$. □

$Q(N)$ is Ramanujan's Q-function, which grows like $\sqrt{\pi N/2} - 1/3$ [49]. The expected fill rate is 0.75, 0.630, 0.555, and 0.406 for $N = 2$, 3, 4, and 8, and decays like $\sqrt{\pi/(2N)}$. Empty slots waste no hashing work (Section 4.5), but they make the tree sparser and deeper than a complete N-ary tree. Slot indices therefore grow faster than the block count (Section 8.3). Network delay has two further effects. It keeps c open longer, which raises the fill rate a little, and it lets blocks arrive after their parent has retired, which wastes work. Section 8.4 measures both.

Table 3: Smallest number k of lane blocks that keeps the reversal probability of Eq. (7) below $\varepsilon = 10^{-3}$, for adversarial share β and N lanes, without network delay. A dash means that β reaches the lane-capture threshold $1/(N+1)$, so no number of confirmations suffices.

β	$N=1$	$N=2$	$N=3$	$N=4$	$N=8$
0.05	4	5	7	9	23
0.10	5	9	15	26	960
0.15	8	17	39	119	–
0.20	11	34	170	–	–

6.4 Lane capture and confirmation depth

Proposition 5 (Lane-capture threshold)**.** *If honest hash power is spread evenly over the N lanes and the adversary concentrates its share β on one lane, it holds the share*

$$\beta_L = \frac{\beta}{\beta + (1-\beta)/N} \tag{6}$$

of the hash power on that lane, and it outpaces the honest miners of the lane if and only if $\beta_L > 1/2$, that is, $\beta > 1/(N+1)$.

The threshold is Nakamoto's $1/2$ for $N = 1$, $1/4$ for $N = 3$, and about 0.11 for $N = 8$. Designs with miner-chosen chains or shards generally face this dilution, and hash-based assignment, merged mining, and adversary-resistant allocation were designed to avoid it [11, 16, 20]. We state it here to make the cost of miner-chosen lanes explicit.

Below the threshold, consider an adversary that mines a private version of a lane to reverse a transaction after k blocks of that lane have confirmed it. If the fork point is recent, blocks of the other lanes can be copied into its tree unchanged. They reference parents older than the fork point (Observation 1), and their lane links do not involve the attacked lane. Section 6.5 discusses the transactions that complicate this copying. The attack is then a race against the honest miners of one lane, and Nakamoto's analysis applies with the lane share β_L [1]. The adversary catches up with probability

$$P(k) = 1 - \sum_{i=0}^{k} \frac{\mu^i e^{-\mu}}{i!} \left(1 - \left(\frac{\beta_L}{1-\beta_L}\right)^{k-i}\right), \qquad \mu = k\,\frac{\beta_L}{1-\beta_L}. \tag{7}$$

Table 3 lists the smallest k with $P(k) < \varepsilon = 10^{-3}$. For $N = 1$ and $\beta \geq 0.10$ it reproduces Nakamoto's figures. With $\beta = 0.10$ and $N = 3$, the adversary holds a quarter of the attacked lane's hash power, which leaves the lane as exposed as a linear chain facing a 25% adversary. The lane then needs 15 confirmations where a linear chain facing the same 10% adversary needs 5, and at $N = 8$ it needs 960. Since each lane produces one block per interval I, confirmation latency grows by the same factor. Parallel lanes thus trade per-lane security for block rate, unless the lanes share hash power.

Equation (7) ignores network delay, so Table 3 is optimistic. Delay-aware analyses of the longest-chain rule require larger depths [7, 50]. If an attacked lane is treated as an isolated chain with honest rate $(1-\beta)\lambda/N$, the condition of Dembo et al. becomes $\beta\lambda < \frac{(1-\beta)\lambda/N}{1+(1-\beta)\lambda\Delta/N}$ [7]. It reduces to Proposition 5 as $\lambda\Delta \to 0$. We use it only as a heuristic, because the lanes are not independent chains.

6.5 Scope of reorganisations

A reorganisation can be narrower than in a linear chain. Suppose a node adopts a tree that differs from its own only in the blocks of lane a created after cycle t_f, and let t be the current cycle. If $t_f > t/(\phi N)$, the blocks of other lanes keep their positions and links. By Observation 1, blocks created after cycle $t/(\phi N)$ have, approximately, no children yet, and the lane links of other lanes do not involve lane a. In a linear chain, every block after the fork point is replaced.

Transactions can still tie lanes together. If a lane-3 transaction spends an output from a replaced lane-1 block, that transaction becomes invalid, and so does its lane-3 block. The reorganisation then spreads to every lane that spent outputs of the replaced blocks. A simple rule bounds this effect: an output may be spent in another lane only after k further blocks of its own lane. A reorganisation shallower than k lane blocks then stays within its lane, and the dependence on sibling arrival order (Section 5.3) disappears. Eunomia considers this rule but, to save latency, admits such spends at once and confines invalidations to a confirmation window [19]. Our prototype does not implement it yet.

6.6 Other attacks and incentives

Timing within the window. A producer that fills a child of c' can publish it at once to close c. Miners still mining for c lose no work, but the remaining slots of c stay empty, and blocks for c still in transit arrive too late. A producer may also skip c and mine under c' before its lane has a child of c. This stays inside the window and leaves the order of parents unchanged, and no node can tell it apart from a lane that found c already closed. Its effects are those of early publication. Honest miners, including those of our prototype, fill c first.

Withholding. Withholding blocks within a lane enables selfish mining at the level of a lane, with the lane share β_L in place of the global share [51]. We do not analyse rational deviations and leave an analysis of optimal strategies, in the style of Sapirshtein et al., to future work [52].

Difficulty and timestamps. All nodes compute the same target under the retarget rule of Section 4.8, but we have not analysed its security when hash power varies. Wang et al. showed that adaptations of Bitcoin's rule to parallel chains can be exploited with forged timestamps [32], and timestamp manipulation has been observed in deployed systems [53]. A deployment needs timestamp validity rules, such as a median-time lower bound and a limit on future drift, and a fork-choice rule that compares accumulated work across retarget periods.

7 Implementation

We implemented the proof-of-work instantiation in Rust, in about 9,800 lines of code and 2,800 lines of tests. The node uses Tokio for networking, Actix for an HTTP interface, and the `k256` crate for ECDSA over secp256k1. A small React front end visualises the tree and the PQP.

Tree store and PQP. Blocks live in an insertion-ordered hash map in which a block's position equals its slot index. Empty slots hold local placeholder records that are never transmitted. This dense layout keeps the prototype simple but does not scale (Section 8.3). A second map lists the children of each block. The PQP module implements Rules 1 and 2 and the search for the lane predecessor. If a block fails a later check, the node restores its PQP to the state before the block.

Miner. The miner of lane a takes its slot from the PQP, selects mempool transactions of lane a, adds a coinbase transaction, signs its PQP entry, and searches for a nonce. It rebuilds its template when a block arrives for its own slot or for the next parent.

Network. Nodes exchange length-prefixed JSON messages over TCP and flood blocks and transactions to all peers. A joining node synchronises through a Bitcoin-style exchange of block locators, inventories, and data requests, and replays the received tree in slot order. Peers find each other through a small registry service, a simplification for experiments. Every 5 s, each node sends its peers its block count and tree hash. When a peer has reported a better tree for 8 s, the node fetches that tree, validates it block by block, and adopts it if it is valid. Transactions of discarded blocks return to the mempool.

Transactions. Transactions follow Bitcoin's format, with pay-to-public-key-hash outputs and m-of-n multisignature outputs behind pay-to-script-hash. The mempool orders transactions by fee rate and supports replacement by fee. A node computes the lane of every input from the locking script it spends and rejects transactions whose inputs fall into different lanes. Validators recompute the lane of every transaction in a block. A node can create its wallet in a chosen lane by drawing keys until one maps there.

Parameters and scope. The default configuration uses $N = 3$, one block per lane every 100 s, a retarget every 300 blocks, a subsidy of 50 coins that halves every 1,000 blocks, a coinbase maturity of 10 slots, and a block limit of 10,240 weight units. Several parts of the design are not yet implemented: smart contracts, the cross-lane spending rules of Sections 5.3 and 6.5, sparse 64-bit slot storage, the mitigations of lane capture, timestamp validity rules, and a check of the proof of work of blocks received from peers. When a node synchronises from scratch, the prototype also accepts a lane link that matches either of two candidate commitments, a looser check than Algorithm 1. None of this affects the experiments below, where all nodes are honest, but the missing checks are needed before the prototype faces an adversary.

Table 4: Setup of the testbed run.

Item	Setting
Hardware	one laptop, 10-core Apple M4, 16 GB memory; nodes connected over the loopback interface; debug build
Nodes and lanes	six nodes, $N = 3$, two miners per lane; five peers per node before mining
Difficulty	$I = 15$ s, $R = 10$, so one block every 5 s; retarget about every 30 blocks (27 to 31 in the run, since a block's count depends on its sibling group)
Workload	every 4 s, each wallet without an unconfirmed payment sends 0.1 to 1 coin, taken from its oldest outputs, to a random other wallet; at most one payment in flight per wallet; two wallets per lane
Measurement	a driver queries every node every 30 s; after 1,500 s of mining it stops the miners, waits 30 s, and saves each node's final state

Table 5: Summary of the 25-minute testbed run ($N = 3$, six nodes).

Metric	Value
Blocks mined (lanes 1/2/3)	258 (81/90/87)
Snapshots with identical block sets	50 of 51
Nodes with identical final trees	6 of 6
Blocks passing slot and lane-link checks	258 of 258
Difficulty retargets	8
Blocks carrying the target prescribed by the rule	258 of 258
Slot races / tree switches	0 / 0
Seconds per block after 600 s (target 5 s)	4.7
Filled slots under retired parents	256 of 393 (65.1%)
Median ratio of parent position to block position (model)	0.50 (0.51)
Transactions in the lane of their inputs	332 of 332
Inclusion latency, median / 90th percentile	13.9 s / 36.1 s

8 Evaluation

We answer the four research questions with three kinds of experiment. A six-node testbed runs the prototype end to end (RQ1, RQ2), and a discrete-event simulation adds the propagation delay that one machine cannot provide (RQ3). A study of 762,842 Bitcoin transactions shows how real owners would spread over lanes (RQ4). RQ1 tests goal G1, RQ2 and RQ3 measure the cost of goal G2, and RQ4 examines goal G3. We do not evaluate goal G4, because we implemented one consensus layer. Section 9 relates the results to measured delays of deployed networks.

8.1 Testbed and workload

All six nodes ran on one laptop (Table 4), two mining in each of $N = 3$ lanes, and every node kept its own tree, PQP, mempool, and wallet. The default configuration retargets only after 300 blocks, which a 25-minute run does not reach. We therefore set $I = 15$ s and $R = 10$, so that the tree aims at one block every 5 s and retargets about every 30 blocks. The workload consists of payments between the six wallets, each of which holds an address in a lane that its own node does not mine, so every payment has to be mined by another node. Table 5 summarises the run.

8.2 RQ1: The nodes agree on the tree and the difficulty

All six nodes held the same block set at 50 of the 51 snapshots. At the remaining snapshot, taken at 1,022 s, node 1 had not yet received the newest block (Figure 4). At the end, every node stored the same tree of 258 mined blocks.

We checked this tree independently of the node software. Every block occupies slot $N\,s(\mathrm{par}(b)) + a(b)$, every lane link commits to the nearest earlier block of its lane, and every block hash, recomputed from its fields, matches and lies below its target.

The prototype's PQP check passed on every node, and its tree check, run again on each node's saved state, accepted all six trees. No miners raced for a slot, no node switched trees, and every node answered every query within 50 ms.

A separate 180 s run tested the fork choice. We cut node 6 off from incoming blocks and tree summaries between 25 s and 85 s, and its tree grew to 37 blocks against 59 elsewhere. Within 35 s after the cut ended, node 6 fetched the longer

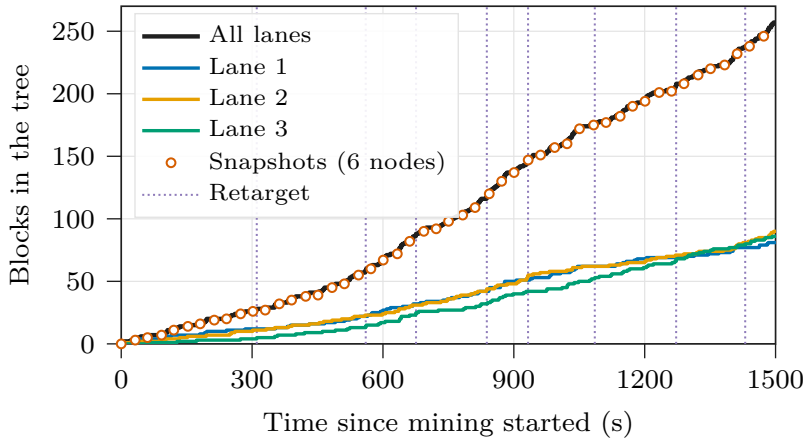


Figure 4: Growth of the tree during the testbed run. Lines count the blocks of each lane and of the whole tree; circles show the block count that node 1 reported every 30 s, which the other five nodes matched at every snapshot except the one at 1,022 s. Dotted lines mark the eight difficulty retargets.

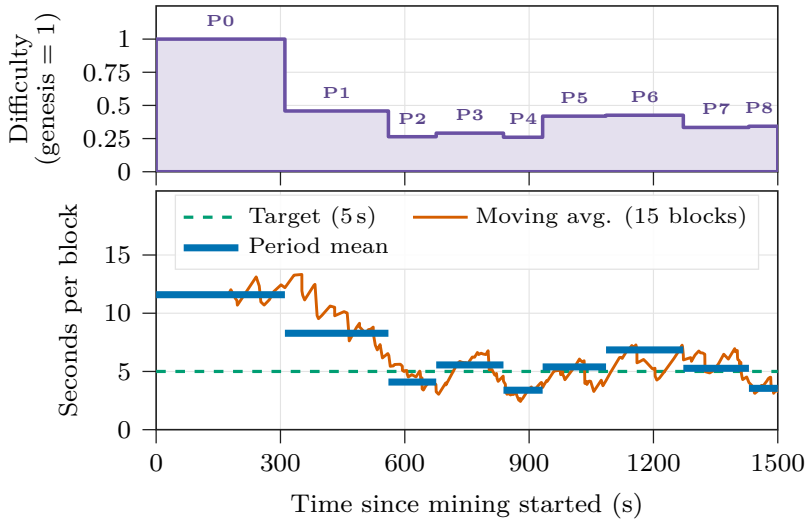


Figure 5: Difficulty adjustment in the testbed run. Top: difficulty of each retarget period P0–P8 relative to the genesis target. Bottom: seconds per block across the whole tree, as a moving average over 15 blocks and as the mean of each period, against the 5 s target.

tree and switched to it, returning the transactions of its discarded blocks to the mempool. One other node switched once to resolve a one-block difference. From 153 s on, all six nodes held the same tree, and all ended with 76 mined blocks.

The nodes also agreed on difficulty. All 258 blocks carry exactly the target that the rule prescribes, as we confirmed by recomputing all nine periods from the final tree. The genesis target was too hard for six miners on one machine: the first 27 blocks arrived every 11.6 s instead of every 5 s. The first two retargets lowered the difficulty to 46% and then 26% of the genesis value (Figure 5). From the third period on, the mean interval stayed between 3.4 and 6.9 s. With about 30 blocks per period, chance alone moves a period's mean by about 18%. Over the last 900 s, the tree produced a block every 4.7 s.

Once, at the start of a period, two lane-3 miners briefly built a template with a different target, because one block of the preceding sibling group had not reached them. Both rebuilt their template when it arrived, so no block carried the wrong value.

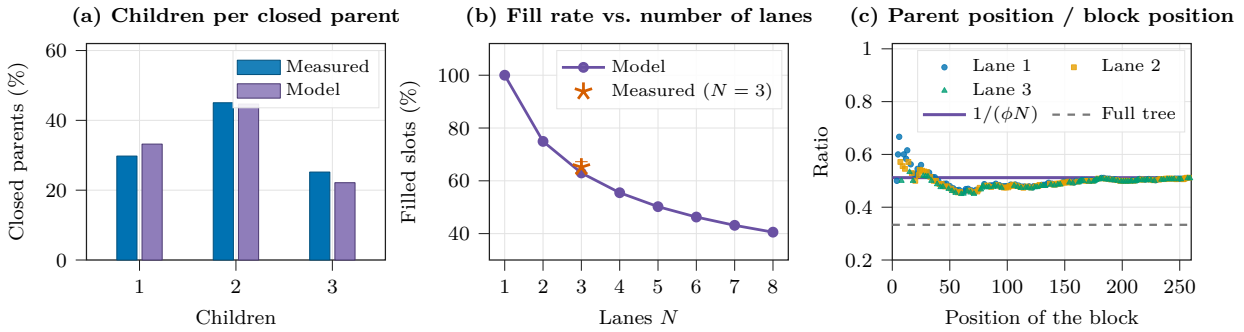


Figure 6: Shape of the tree in the testbed run. (a) Number of children of the 131 parents retired during the run, compared with the model of Proposition 4. (b) Fill rate for $N = 1, \ldots, 8$ in the model and the value measured at $N = 3$, with one standard deviation of the model for 131 parents. (c) Position of each block's parent divided by the position of the block, with blocks numbered in slot order; the solid line is the prediction $1/(\phi N)$ for the measured fill rate $\phi = 0.651$, and the dashed line is the value $1/N$ of a full tree.

8.3 RQ2: The window model predicts the shape of the tree

The measured fill rate was 65.1%: 256 of the 393 slots under the 131 retired parents. Proposition 4 predicts 63.0%, with a standard deviation of 2.1 points for 131 parents, so the measurement lies about one standard deviation above the model (Figure 6b). The distribution of children agrees too. Of the retired parents, 30%, 45%, and 25% received one, two, and three children, against 33%, 44%, and 22% in the model (Figure 6a).

Empty slots stretch the tree. After 258 blocks the highest slot index was 5,739, so each node stored about 5,500 placeholder records. At this fill rate, 32-bit slot indices would run out at depth 20 or 21, after a few hundred thousand to about a million blocks. A deployment therefore needs sparse storage with 64-bit indices, which the prototype does not yet have.

Parent references point far back, as Observation 1 predicts. With the blocks numbered in slot order, the parent of the block at position p should sit near position $p/(\phi N)$. The median ratio was 0.50, against $1/(\phi N) = 0.51$ for the measured $\phi = 0.651$ and $1/3$ for a full tree (Figure 6c). A new block thus points to a parent created about halfway back in the history of the tree. Until it becomes a parent itself, only later blocks of its lane protect it.

8.4 RQ3: Under delay, DetTreeChain wastes less work than a chain at the same block rate

In simulation, at a delay of 5% of the lane interval, DetTreeChain wasted 4.7% to 8.3% of mined blocks for $N = 2$ to 8, and a single chain producing blocks at the same total rate wasted 1.7 to 3.3 times as much (Table 6, Figure 7). At a delay of 20% the factor narrows to 1.6 to 2.3. We used simulation because blocks in our testbed propagated almost instantly, and none arrived after its parent had retired.

Simulation model. The discrete-event simulator implements Rules 1 and 2, the miners' choice of slot, and first-seen acceptance. It uses four miners per lane with equal hash power and a total block rate of N/I. In a full mesh, each block reaches each node after a fixed delay Δ, or after an exponential delay with mean Δ; here Δ is the actual delay, whereas Section 3 uses it as a bound. A node adopts a peer's tree when it holds strictly more blocks, and a block counts as wasted if it is not in the final tree. For $N \in \{1, 2, 3, 4, 8\}$ and $\Delta/I \in \{0.01, 0.02, 0.05, 0.1, 0.2\}$ we ran five runs of 50,000 blocks, which keeps every standard error below 0.14 percentage points. The two baselines are single chains, one with block rate N/I and one with interval I and N times larger blocks, modelled as delay $N\Delta$. Without delay, the simulator reproduces Proposition 4.

Where the waste comes from. Table 6 splits the waste into its two sources. Miners of one lane racing for a slot cost about 3.5% of mined blocks at $\Delta/I = 0.05$, for every N, as in a single chain with interval I. The window adds a second source: a block for c that arrives after the first child of c'. Its probability grows with the number of lanes already working under c' when a child of c is mined. Because c closes at the first repeated lane (Section 6.3), this number stays small: averaged over all mined blocks at $\Delta/I = 0.05$, counting zero for blocks mined under c', it is 0.33, 0.58, 0.79, and 1.43 for $N = 2$, 3, 4, and 8. The window term, 1.0% to 4.3% of mined blocks, is the price of bounding the drift. Parallel chains that never wait for one another, which we did not simulate, would in this model waste roughly what one chain with interval I wastes, 3.6% at $\Delta/I = 0.05$, but would leave the order of their blocks to be settled later.

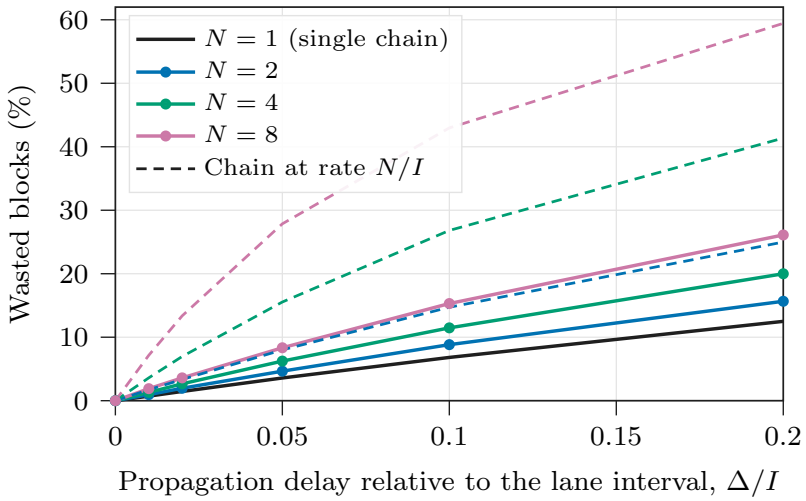


Figure 7: Share of mined blocks that end up outside the final tree, as a function of the propagation delay Δ relative to the lane interval I (fixed delay, five runs of 50,000 blocks per point; standard errors below 0.14 points). Solid lines show DetTreeChain with N lanes; dashed lines show a single chain producing blocks at the same total rate N/I.

Table 6: Simulation at $\Delta/I = 0.05$ (fixed delay): wasted blocks of DetTreeChain split into slot races and window losses (blocks whose parent the final tree had already retired or not yet opened; the rest of the total are blocks that built on a discarded block), wasted blocks of a single chain at the same total rate, and the share of nodes whose tree held a block that was later discarded. All values in %.

	DetTreeChain, wasted			Chain at rate N/I,	Nodes holding a discarded block	
N	Races	Window	Total	wasted	DetTreeChain	Chain
1	3.5	–	3.6	–	1.6	–
2	3.6	1.0	4.7	8.0	3.2	2.1
3	3.6	1.7	5.5	11.9	5.0	2.2
4	3.6	2.3	6.2	15.5	6.7	2.3
8	3.4	4.3	8.3	27.9	10.8	2.6

Baselines and delay distributions. The chain with larger blocks behaved like the faster chain, within 0.3 points. This is expected by construction, since both have the same product of block rate and delay, so the second baseline adds no independent evidence. This baseline assumes that delay grows in proportion to block size, as Decker and Wattenhofer observed for Bitcoin blocks larger than 20 kB [3]. Where latency dominates instead of bandwidth, as Kiffer et al. found for Ethereum [54], larger blocks would waste less than this baseline suggests and would be limited by node bandwidth instead (Section 9.2). With exponential delays, DetTreeChain wasted less than with fixed delays: 5.4% at $N = 8$ and $\Delta/I = 0.05$, against 23.5% for the chain. Delay also kept the current parent open longer, and the fill rate at $N = 8$ rose from 40.7% without delay to 46.4% at $\Delta/I = 0.2$.

The cost: more frequent disagreement. A late block stays in its miner's view until later blocks of its lane settle the fork, so nodes disagreed more often than in a single chain. At $N = 8$ and $\Delta/I = 0.05$, at least two nodes held different trees during 73.9% of the simulated time, against 52.1% for the chain, and 97.7% against 91.0% at 0.2. This measure also counts blocks still in transit. A sharper one is the share of nodes whose tree held a block that was later discarded: 10.8% against 2.6% at $\Delta/I = 0.05$, and 21.5% against 10.9% at 0.2. A block that a node has just accepted is thus more likely to be reverted than in a single chain, and users should wait for later blocks of its lane.

Growth with delay. For small delays (least-squares slope through the origin for $\Delta/I \leq 0.02$), each percentage point of Δ/I added 0.98 to 1.82 percentage points of waste to DetTreeChain for $N = 2$ to 8, against 1.68 to 6.78 for the faster chain and 0.73 for a single chain with interval I. The waste of DetTreeChain thus grows with delay roughly as one lane's does, plus a window term that grows with N, whereas a single chain's waste grows with its full block rate. The model has no adversary, no bandwidth limit, and a simplified fork choice, so these results show trends and should not be read as deployment figures.

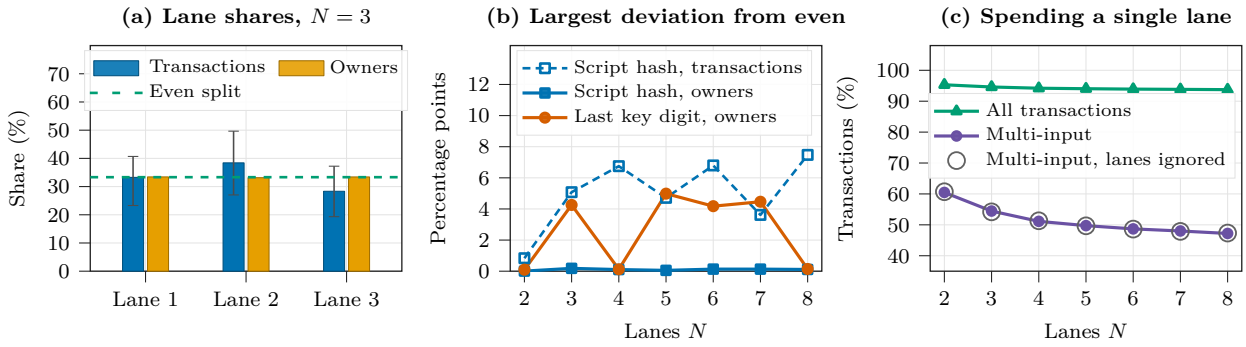


Figure 8: Routing applied to transactions from 200 random Bitcoin blocks (September 2025 to September 2026), with lanes given by the hash of the locking script. (a) Lane shares for $N = 3$, counted by transactions and by owners (distinct locking scripts); whiskers span the 10th to 90th percentile of the transaction share over single blocks. (b) Largest deviation of any lane from an even split, for the script hash and, for comparison, for the last hexadecimal digit of the public key. (c) Share of transactions whose inputs all lie in one lane, over all transactions and over those with several inputs, compared with the share expected if each locking script were assigned to a lane at random.

8.5 RQ4: Real owners spread evenly, and most transactions already fit one lane

Owners of real Bitcoin transactions spread evenly over lanes, although a few busy owners carry much of the load, and 94.6% of the transactions we could analyse would already be valid at $N = 3$. The testbed checks routing end to end, and the Bitcoin study estimates how real owners would use it.

Routing in the testbed. Every confirmed transaction landed in the lane of its inputs, including the 86 with several inputs. Most payments crossed lanes: 267 of the 332 confirmed payments created an output in another lane. Later, 125 of those outputs were spent, each in the lane of its locking script, without any message between lanes. The wallets submitted 497 payments, and all 163 rejections came in the first 169 s, before the wallets held mature coins. The median time from submission to inclusion was 13.9 s and the 90th percentile 36.1 s, about one and two and a half lane intervals. Blocks carried 1.29 user transactions on average against a capacity of about 11.

Bitcoin data. Since six wallets say little about real owners, we took 200 blocks chosen at random from the year ending at height 968,126 (September 2025 to September 2026) and checked each against its header hash and Merkle root. For each of the 762,842 non-coinbase transactions, we rebuilt the locking scripts of the spent outputs from what the inputs reveal. These are the public key of a P2PKH or P2WPKH spend and the redeem or witness script of a P2SH or P2WSH spend. For two inputs of each of the six input types, all 12 rebuilt scripts matched the actual outputs. Taproot spends and a few rare types reveal no script, so the analysis covers the 635,952 transactions (83.4%) whose first input does. A node reads scripts from its UTXO set and has no such limit. We count each of the 253,292 distinct locking scripts as one owner, although one person often controls many scripts [48].

Owners spread evenly. At $N = 3$ the lanes hold 33.4%, 33.2%, and 33.4% of the owners. For every N from 2 to 8, no lane deviates from an even split by more than 0.2 percentage points (Figure 8a,b). A rule based on the last hexadecimal digit of the key, used by an earlier version of our prototype, is biased whenever N does not divide 16. At $N = 3$, its first lane receives the digits 0, 3, 6, 9, c, and f and holds 37.6% of the keys.

Load concentrates on a few owners. Transactions spread less evenly, because a few owners are very active. The busiest 1% of owners appear in 58% of the transactions, and one locking script appears in 22,343 of them, while 94% of owners appear only once. At $N = 3$ the lanes receive 33.3%, 38.4%, and 28.3% of the transactions, so the busiest lane carries 1.15 times its fair share. The imbalance grows with N: at $N = 8$ the busiest lane receives 20.0% of the transactions, 1.6 times its fair share. Within single blocks, a lane's share ranges from about 19% to 50% between the 10th and 90th percentiles. Sharded systems face the same problem with hot accounts [34, 38].

Most transactions already satisfy the routing rule. Of the transactions, 88% spend a single output and satisfy Rule 3 trivially. Among transactions with several inputs, 44% spend outputs of one locking script, and at $N = 3$, 54% have all inputs in one lane. This matches the share expected if each locking script were assigned to a lane at random (Figure 8c). Even though today's wallets ignore lanes, 94.6% of transactions would be valid at $N = 3$. This

share counts single-input transactions that reveal their script and multi-input transactions whose inputs all reveal theirs. Across N from 2 to 8 the share lies between 93.7% and 95.3%, so 4.7% to 6.3% are cross-lane transactions. These would have to be split, or be created by lane-aware wallets (Section 5.4).

These figures should not be compared directly with cross-shard ratios under state sharding, such as the 93.5% that OptChain measured for random placement over 16 shards [36]. There, a transaction is cross-shard when its inputs or its outputs lie in different shards. In DetTreeChain, outputs may go to any lane, because every node stores all of them.

8.6 Threats to validity

Internal validity. All testbed nodes shared one machine. Blocks propagated almost instantly, and the six miners shared ten cores, so their hash rates were neither equal nor independent. Unequal hash rates may explain why the fill rate lay one standard deviation above the model. The difficulty parameters were scaled down, and the nodes ran a debug build, so absolute timings are not representative. The prototype does not yet check the proof of work of received blocks or validate timestamps. These gaps were harmless here because every node was honest, and our offline check confirmed a valid proof of work for every block.

Construct validity. The fill rate and the share of wasted blocks measure how the window shapes the tree and say nothing about transaction throughput. The six-wallet workload was far too small to fill the blocks, so we make no throughput claim. In the simulation, waste is counted against the final tree of a model with a simplified fork choice.

External validity. The simulation assumes a full mesh, independent delays, equal hash power, four miners per lane, and no bandwidth limit, so a block's delay depends neither on its size nor on how many other blocks are in flight. Concurrent lane blocks that share links would propagate more slowly than modelled. The absolute numbers of the simulation are likely to differ on a real network. The Bitcoin study covers 200 blocks from one year and the 83.4% of transactions whose first input reveals its script. Owners are locking scripts, which often differ from people, and lane-aware wallets would behave differently from today's.

9 Discussion

9.1 What the results mean for deployed networks

How large is Δ/I in practice? For Bitcoin in 2012, Decker and Wattenhofer measured a median block propagation delay of 6.5 s and a mean of 12.6 s [3]. With a 600 s interval, this gives Δ/I of about 0.01 to 0.02. Propagation has since become faster, and Bitcoin's stale rate fell to 0.41% around 2016 [4], so the 2012 figures are a pessimistic setting.

Ethereum's proof-of-work chain had an interval of about 13 s and block propagation delays, measured from the first observation at the authors' vantage points, of 74 ms at the median and 317 ms at the 99th percentile [55]. Its stale rate, however, was 6.4% to 7.2% [54, 55], far above what such delays imply. One plausible reason, which we have not tested, is that these measurements start at the first observing node and miss the time from the miner to the network. In our model, a 7% stale rate for a single chain corresponds to an effective delay of about 1.4 s, or $\Delta/I \approx 0.1$.

Table 7 maps these settings onto the simulation by linear interpolation between grid points. At Bitcoin-like delays, eight lanes would produce eight blocks per ten minutes and waste about 1 to 2 percentage points more than one chain, but only about a quarter of what a chain running eight times faster wastes. With exponential delay the waste of eight lanes falls to 1.3% to 2.4%. This assumes that each lane block propagates as fast as one 2012 Bitcoin block; if delay grows with block size, lane blocks must be correspondingly smaller (Section 9.2). At the effective Ethereum delay, eight lanes waste 15.3%, twice as much as one chain and a third as much as the faster chain. The window still helps there, but the absolute waste is high.

These numbers indicate trends only. The single-chain model matches Bitcoin's 2012 fork rate of 1.69% [3] for an effective delay of about 14 s, close to the measured mean, but gives 0.8% for the median delay. For Ethereum, the measured inter-node delays give 0.4% to 1.7%, well below the observed stale rate. The mapping from measured delay to Δ is therefore uncertain by a factor of two or more. The ratio between the waste of eight lanes and that of the faster chain is more stable: about 0.27 for all four measured delays and 0.36 at the effective Ethereum delay.

9.2 What the lanes scale, and what they do not

Every node receives, stores, and validates the blocks of all lanes, so lanes leave the per-node throughput bound unchanged. Croman et al. estimated that, at 2015 node bandwidths, a ten-minute interval allows blocks of at most 4 MB,

Table 7: Wasted blocks (%) in the simulation model for delays measured in deployed networks (fixed delay, linear interpolation between grid points). The delays are node-level measurements, so the mapping is approximate; "effective" is the delay at which the single-chain model matches the measured stale rate.

Setting	Δ	I	Δ/I	Measured stale	One chain	DetTreeChain, $N = 8$	Chain, $8\times$ rate
Bitcoin 2012, median [3]	6.5 s	600 s	0.011	1.69	0.8	2.0	7.7
Bitcoin 2012, mean [3]	12.6 s	600 s	0.021	1.69	1.5	3.8	13.9
Ethereum 2019, median [55]	0.074 s	13.3 s	0.006	6.4–7.2	0.4	1.1	4.0
Ethereum 2019, 99th pct. [55]	0.317 s	13.3 s	0.024	6.4–7.2	1.7	4.2	15.2
Ethereum, effective	1.4 s	13.3 s	0.10	6.4–7.2	6.8	15.3	43.0

Table 8: Ways to produce eight blocks per ten minutes with $\Delta = 12.6$ s: wasted blocks in the simulation model (fixed delay, interpolated) and confirmations against a 10% adversary from Table 3. The confirmation counts ignore delay, which favours the short intervals.

N	I	Δ/I	Wasted (%)	k at $\beta = 0.1$	Time to reach k
1	75 s	0.168	10.7	5	6 min
2	150 s	0.084	7.5	9	23 min
4	300 s	0.042	5.3	26	2.2 h
8	600 s	0.021	3.8	960	6.7 days

about 27 transactions per second, if 90% of nodes are to keep up [2]. The same budget applies to the sum of all lanes. With eight lanes, for example, it would allow lane blocks of about 0.5 MB.

What lanes change is how this budget is spent: as N smaller blocks produced independently instead of one large block. Whether these also reach other nodes sooner depends on whether propagation is bound by latency or by bandwidth, which our simulation does not model. Longest-chain protocols also need care when nodes run close to their bandwidth [21, 22]. Combining lanes with state sharding would lower the load per node, but cross-lane payments would then need messages again.

9.3 Security per lane and the choice of N

With miner-chosen lanes, the higher block rate costs security per lane. For Bitcoin-like lanes, a lane at $N = 3$ needs 15 confirmations against a 10% adversary (Table 3). At 600 s per lane block, that is about 2.5 hours, against 50 minutes for Bitcoin and about 17 minutes for a single chain with the same total block rate. Delay, which this calculation ignores, would raise the last figure. At $N = 8$ a lane needs 960 confirmations, more than six days.

The number of lanes therefore trades wasted work against per-lane security (Table 8). For a fixed total rate of eight blocks per ten minutes and $\Delta = 12.6$ s, the model favours more lanes with longer intervals: 3.8% of blocks are wasted with eight 600 s lanes, 5.3% with four 300 s lanes, and 10.7% with one 75 s chain. Each extra lane, however, lowers the capture threshold and lengthens the time to a safe confirmation depth. Under miner-chosen lanes, the table suggests stopping at a small N. Two 150 s lanes already waste about 30% less than one 75 s chain and reach a safe depth within half an hour, whereas going from four lanes to eight saves only 1.5 points of waste but stretches the time to a safe depth from about two hours to almost a week.

We kept miners' choice of lane deliberately, since it may let hash power follow demand: a lane whose backlog raises its fees should attract miners. We have not evaluated this effect. A deployment that must tolerate a concentrating adversary needs a remedy. With hash-based assignment, as in OHIE and Eunomia [11, 19], each miner would prepare a candidate block for every lane and let the hash decide. In DetTreeChain each candidate carries its own slot, parent, lane link, and transactions, so block format, mining, and validation would all change. Merged mining in the style of Chu-ko-nu mining [16] or Manifoldchain's shared mining power [56], references between lanes in the style of Chainweb [29], and stake-based slot leaders are further options, each to be analysed together with difficulty adjustment [32]. A formal proof of common prefix, chain quality, and chain growth for lane chains [5] also remains open, and ledger combiners [12] are a natural starting point.

9.4 Load balance and privacy

Routing by owner keeps all transactions of a hot owner in one lane. In our Bitcoin sample, the busiest lane carried 1.15 times its fair share at $N = 3$ and 1.6 times at $N = 8$ (Section 8.5). If all lanes have the same capacity and this mix

persisted, the busiest lane would fill first, at about 63% of the total nominal capacity for $N = 8$ and 87% for $N = 3$; bursts within single blocks would fill it earlier. A single very active owner is limited to the throughput of one lane. An owner whose lane is congested can move simply by sending its coins to an address in a lighter lane, a wallet-level form of the account migration that sharded systems use [38]. Lane-aware wallets also leak information, because all their addresses share a lane. Adapting N to the load and quantifying this leak are open questions.

10 Conclusion

DetTreeChain fixes the position of every concurrent block before it is produced. Slots follow from parent and lane, a locally derived window keeps all lanes within one parent of each other, lane links make each lane a hash chain, and routing by locking script keeps conflicting transactions in one lane. The structure of the ledger is thus a function of its blocks, and up to N blocks can be produced per parent cycle without messages between lanes.

Our results suggest that, because blocks of different lanes do not compete for the same position, the lanes absorb much of the concurrency of a high block rate. The window that bounds their drift is paid for mostly in empty slots, which cost no mining work, and partly in late blocks that are discarded. In our runs, all nodes ended with the same tree and difficulty, and the fill rate stayed close to a birthday-problem model. In simulation with delays of 5% of the lane interval, a single chain at the same block rate wasted 1.7 to 3.3 times as much work, although nodes more often held blocks that were later discarded. For delays measured in Bitcoin, the model suggests a few per cent of wasted blocks for eight ten-minute lanes, a figure uncertain by a factor of two or more. Routing by owner fits today's usage, since about 95% of the Bitcoin transactions we analysed already kept their inputs in one lane, but it concentrates load on a few busy owners.

Under proof of work, each lane is protected only by its own hash power: at $N = 3$, a lane needs 15 confirmations against a 10% adversary, where Bitcoin needs 5. Whether hash-based lane assignment or stake-based leaders can remove this cost while keeping the window simple is the main open question.

Acknowledgements

During the preparation of this work, the authors used the generative AI tools ChatGPT (OpenAI) and Claude (Anthropic) to assist with searching and organising the literature, with writing scripts for the simulation and data analysis, and with drafting and language editing of the manuscript. The authors checked every reference against its source, verified all results and statements against the data and the code, and take full responsibility for the content of this article.

Conflict of interest

The authors declare no conflict of interest.

Funding

This research received no external funding.

Data availability statement

The source code of the prototype, the data and logs of the testbed runs, and the list of Bitcoin blocks used in Section 8.5 are available at `https://github.com/DetTreeChain/DetTreeChain_Pow_Model`. The simulator used in Section 8.4 and its results are available in the same repository, in the folder `experiments/delay_simulation`.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. White paper. `https://bitcoin.org/bitcoin.pdf`, 2008. URL `https://bitcoin.org/bitcoin.pdf`. Accessed October 6, 2026.

[2] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains: (a position paper). In *Financial Cryptography and Data Security: FC 2016 International Workshops*, pages 106–125. Springer Berlin Heidelberg, 2016. doi:10.1007/978-3-662-53357-4_8.

[3] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, September 2013. doi:10.1109/p2p.2013.6688704.

[4] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Čapkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, October 2016. doi:10.1145/2976749.2978341.

[5] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology – EUROCRYPT 2015*, pages 281–310. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-46803-6_10.

[6] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673. Springer International Publishing, 2017. doi:10.1007/978-3-319-56614-6_22.

[7] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and Nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878. ACM, October 2020. doi:10.1145/3372297.3417290.

[8] 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. ACM, November 2019. doi:10.1145/3319535.3363213.

[9] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM GHOSTDAG: a scalable generalization of Nakamoto consensus. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 57–70. ACM, September 2021. doi:10.1145/3479722.3480990.

[10] Chenxing 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 *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 515–528. USENIX Association, 2020. URL https://www.usenix.org/conference/atc20/presentation/li-chenxing.

[11] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. OHIE: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105. IEEE, May 2020. doi:10.1109/sp40000.2020.00008.

[12] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In *Theory of Cryptography (TCC 2020)*, pages 322–352. Springer International Publishing, 2020. doi:10.1007/978-3-030-64375-1_12.

[13] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016. doi:10.1145/2976749.2978389.

[14] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018. doi:10.1109/sp.2018.000-5.

[15] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018. doi:10.1145/3243734.3243853.

[16] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 95–112, Boston, MA, February 2019. USENIX Association. URL https://www.usenix.org/conference/nsdi19/presentation/wang-jiaping.

[17] Sébastien Forestier, Damir Vodenicarevic, and Adrien Laversanne-Finot. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. arXiv preprint arXiv:1803.09029, 2020. URL https://arxiv.org/abs/1803.09029.

[18] The ZILLIQA Team. The ZILLIQA technical whitepaper. Version 0.1, https://docs.zilliqa.com/whitepaper.pdf, August 2017. Accessed October 6, 2026.

[19] Jianyu Niu. Eunomia: A permissionless parallel chain protocol based on logical clock. arXiv preprint arXiv:1908.07567, 2019. URL https://arxiv.org/abs/1908.07567.

[20] Ranvir Rana, Sreeram Kannan, David Tse, and Pramod Viswanath. Free2Shard: Adversary-resistant distributed resource allocation for blockchains. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–38, 2022. doi:10.1145/3508031.

[21] Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse, and Mohammad Alizadeh. Longest chain consensus under bandwidth constraint. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 126–147. ACM, September 2022. doi:10.1145/3558535.3559777.

[22] Lucianna Kiffer, Joachim Neu, Srivatsan Sridhar, Aviv Zohar, and David Tse. Nakamoto consensus under bounded processing capacity. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 363–377. ACM, December 2024. doi:10.1145/3658644.3670347.

[23] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59. USENIX Association, 2016. URL https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal.

[24] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in Bitcoin. In *Financial Cryptography and Data Security (FC 2015)*, pages 507–527. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-47854-7_32.

[25] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *Financial Cryptography and Data Security (FC 2015)*, pages 528–547. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-47854-7_33.

[26] Patrik Keller, Ben Glickenhaus, George Bissias, and Gregory Griffith. Tailstorm: A secure and fair blockchain for cash transactions. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, pages 6:1–6:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.AFT.2023.6.

[27] Ali Dorri and Raja Jurdak. Tree-Chain: A fast lightweight consensus algorithm for IoT applications. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 369–372. IEEE, November 2020. doi:10.1109/lcn48667.2020.9314831.

[28] Ali Dorri and Raja Jurdak. Tree-Chain: A lightweight consensus algorithm for IoT-based blockchains. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, May 2021. doi:10.1109/icbc51069.2021.9461098.

[29] Will Martino, Monica Quaintance, and Stuart Popejoy. Chainweb: A proof-of-work parallel-chain architecture for massive throughput. Kadena white paper, draft v15. Archived copy: https://web.archive.org/web/20250414235356/https://d31d887a-c1e0-47c2-aa51-c69f9f998b07.filesusr.com/ugd/86a16f_029c9991469e4565a7c334dd716345f4.pdf, 2018. Accessed November 9, 2026.

[30] Lucianna Kiffer, Rajmohan Rajaraman, and abhi shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 729–744. ACM, October 2018. doi:10.1145/3243734.3243814.

[31] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology – CRYPTO 2017*, pages 291–323. Springer International Publishing, 2017. doi:10.1007/978-3-319-63688-7_10.

[32] Xuechao Wang, Viswa Virinchi Muppirala, Lei Yang, Sreeram Kannan, and Pramod Viswanath. Securing parallel-chain protocols under variable mining power. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1700–1721. ACM, November 2021. doi:10.1145/3460120.3485254.

[33] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. SoK: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61. ACM, 2019. doi:10.1145/3318041.3355457.

[34] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 1968–1977. IEEE, 2022. doi:10.1109/infocom48880.2022.9796859.

[35] Yibin Xu, Jingyi Zheng, Boris Düdder, Tijs Slaats, and Yongluan Zhou. A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance. In *Proceedings 2024 Network and Distributed System Security Symposium*. Internet Society, 2024. doi:10.14722/ndss.2024.24006.

[36] Lan N. Nguyen, Truc D. T. Nguyen, Thang N. Dinh, and My T. Thai. OptChain: Optimal transactions placement for scalable blockchain sharding. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 525–535. IEEE, 2019. doi:10.1109/icdcs.2019.00059.

[37] Yuanzhe Zhang, Shirui Pan, and Jiangshan Yu. TxAllo: Dynamic transaction allocation in sharded blockchain systems. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 721–733. IEEE, 2023. doi:10.1109/icde55515.2023.00390.

[38] Mingzhe Li, Wei Wang, and Jin Zhang. LB-Chain: Load-balanced and low-latency blockchain sharding via account migration. *IEEE Transactions on Parallel and Distributed Systems*, 34(10):2797–2810, 2023. doi:10.1109/tpds.2023.3238343.

[39] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 307–316. ACM, 2019. doi:10.1145/3293611.3331589.

[40] Mathieu Baudet, George Danezis, and Alberto Sonnino. FastPay: High-performance Byzantine fault tolerant settlement. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 163–177. ACM, 2020. doi:10.1145/3419614.3423249.

[41] Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, Brandon Williams, and Lu Zhang. Sui Lutris: A blockchain combining broadcast and consensus. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2606–2620. ACM, 2024. doi:10.1145/3658644.3670286.

[42] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 232–244. ACM, 2023. doi:10.1145/3572848.3577524.

[43] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

[44] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

[45] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed BFT. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 1–23. ACM, 2024. doi:10.1145/3694715.3695942.

[46] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017. doi:10.1145/3132747.3132757.

[47] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

[48] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013. doi:10.1145/2504730.2504747.

[49] Philippe Flajolet, Peter J. Grabner, Peter Kirschenhofer, and Helmut Prodinger. On Ramanujan's Q-function. *Journal of Computational and Applied Mathematics*, 58(1):103–116, 1995. doi:10.1016/0377-0427(93)E0258-N.

[50] Peter Gaži, Ling Ren, and Alexander Russell. Practical settlement bounds for proof-of-work blockchains. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1217–1230. ACM, November 2022. doi:10.1145/3548606.3559368.

[51] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security (FC 2014)*, pages 436–454. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-45472-5_28.

[52] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in Bitcoin. In *Financial Cryptography and Data Security (FC 2016)*, pages 515–532. Springer Berlin Heidelberg, 2017. doi:10.1007/978-3-662-54970-4_30.

[53] Aviv Yaish, Gilad Stern, and Aviv Zohar. Uncle Maker: (time)stamping out the competition in Ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–149. ACM, November 2023. doi:10.1145/3576915.3616674.

[54] Lucianna Kiffer, Asad Salman, Dave Levin, Alan Mislove, and Cristina Nita-Rotaru. Under the hood of the Ethereum gossip protocol. In *Financial Cryptography and Data Security (FC 2021)*, pages 437–456. Springer Berlin Heidelberg, 2021. doi:10.1007/978-3-662-64331-0_23.

[55] Paulo Silva, David Vavricka, Joao Barreto, and Miguel Matos. Impact of geo-distribution and mining pools on blockchains: A study of Ethereum. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 245–252. IEEE, June 2020. doi:10.1109/dsn48063.2020.00041.

[56] Chunjiang Che, Songze Li, and Xuechao Wang. Manifoldchain: Maximizing blockchain throughput via bandwidth-clustered sharding. In *Proceedings 2025 Network and Distributed System Security Symposium*, NDSS 2025. Internet Society, 2025. doi:10.14722/ndss.2025.230028.