LazyLedger

Mustafa Al-Bassam

Agenda

- **Part 1: LazyLedger**
- **Part 2: Data availability proofs**
- **Discussion**

What is LazyLedger?

- An alternative design paradigm for the base layer of blockchains.
- Preprint on arXiv: https://arxiv.org/abs/1905.09274

This document is a draft; feedback is welcome.

LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts

Mustafa Al-Bassam[1]

[1]Department of Computer Science
University College London
m.albassam@cs.ucl.ac.uk

Abstract

We propose LazyLedger, a design for distributed ledgers where the blockchain is optimised for solely ordering and guaranteeing the availability of transaction data. Responsibility for executing and validating transactions is shifted to only the clients that have an interest in specific transactions relating to blockchain applications that they use. As the core function of the consensus system of a distributed ledger is to order transactions and ensure their availability, consensus participants do not necessarily need to be concerned with the contents of those transactions. This reduces the problem of block verification to data availability verification, which can be achieved probabilistically with sub-linear complexity, without downloading the whole block. The amount of resources required to reach consensus can thus be minimised, as transaction validity rules can be decoupled from consensus rules. We also implement and evaluate several example LazyLedger applications, and validate that the workload of clients of specific applications does not significantly increase when the workload of other applications that use the same chain increase.

1 Introduction

So far, blockchain-based distributed ledger platforms such as Bitcoin [1] and Ethereum [2] have adopted similar consensus design paradigms, where the validity of the blocks proposed by block producers is determined by (*i*) whether it is the block producer's turn to propose a block and (*ii*) whether the transactions in the block are valid according to some state machine. Traditional consensus protocols such as Practical Byzantine Fault Tolerance [3] have also taken a similar approach, where consensus nodes (replicas) process transactions according to a state machine.

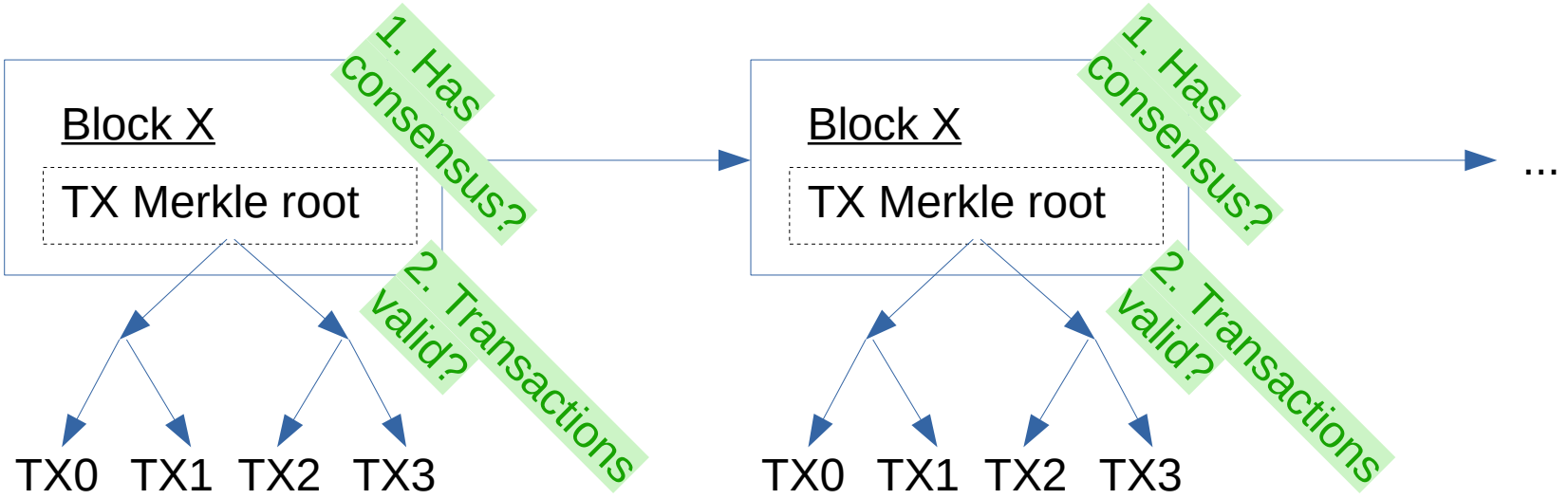
The scalability issues that have plagued decentralised blockchains [4] can be attributed to the fact that in order to run a node that validates the blockchain, the node must download, process and validate every transaction included in the chain. As a result, various scalability efforts have emerged including on-chain scaling via sharding [5, 6], which aims to split the state of the blockchain into multiple shards so that transactions can be processed by different consensus groups in parallel, and off-chain scaling via state channels [7, 8], which takes the approach of moving transactions off-chain and using the blockchain as a settlement layer.

However, it is also worth exploring alternative blockchain design paradigms that may be suitable for different types of applications, where nodes that need to validate the blockchain in order to determine the correct chain do not need to validate the contents of

1

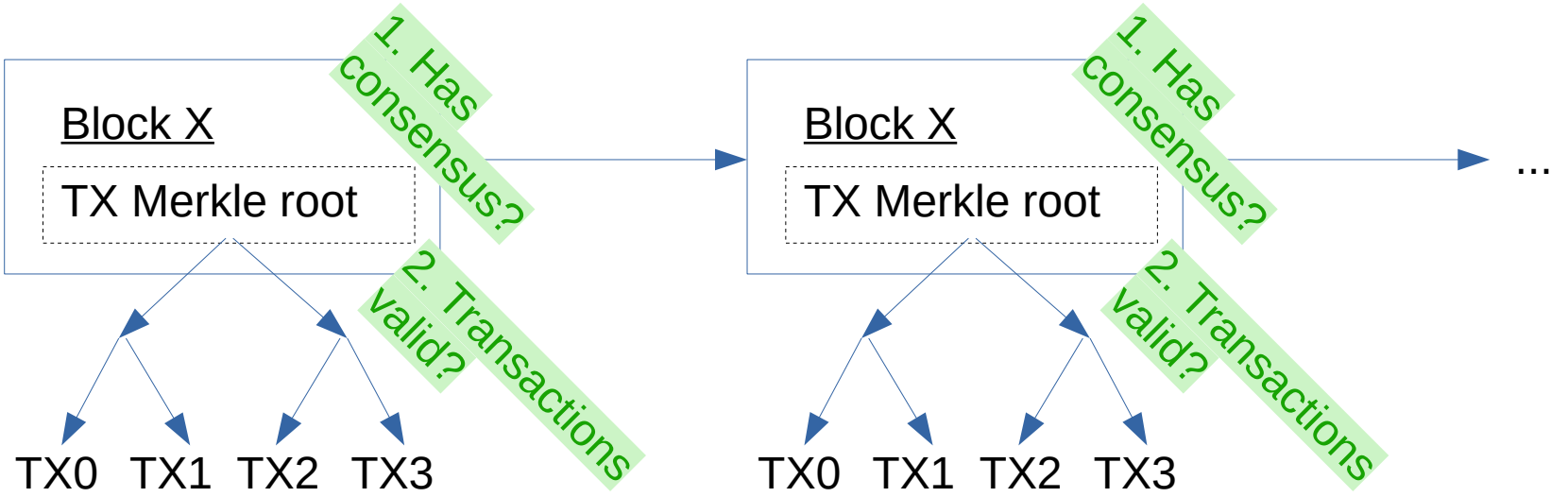
Current blockchain designs

- **Currently, blockchains couple consensus with state execution.**
  - In order to know if a block is valid, you need to:
    1) check that the block has consensus; **and**
    2) check that the transactions in the block are valid.



Current blockchain designs

- Problem: every full node has to download and **execute** every transaction to make sure that blocks are valid.
  - There is an ~O(n) cost to validate a block, where n = size of the block.



LazyLedger design

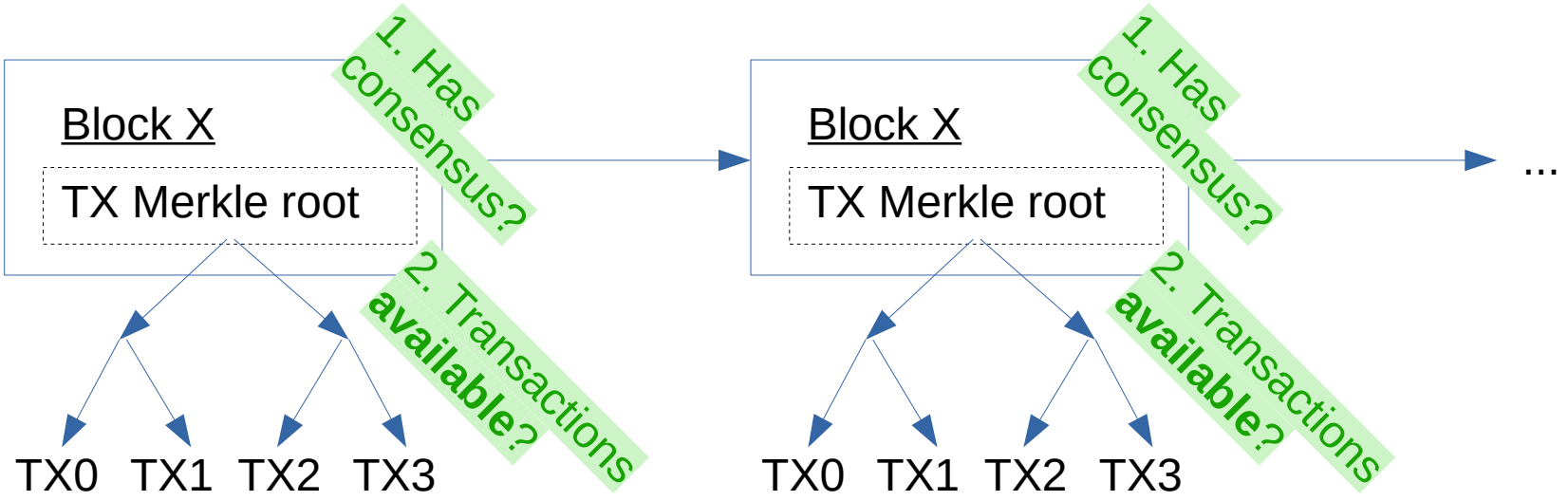
- What if we people were allowed to post **invalid** transactions on-chain, and full nodes don’t need to execute them?
- Nodes can download these transactions, and compute the state of the chain **locally**.
  - Invalid transactions do not change the state.
    - e.g. If there’s a double spend, the second transaction is silently discarded.

LazyLedger design

- **StateTransition(S, T) → S’**
  - The state transition function cannot return an error!
- **Example:**
  - At state S0, TX0 sends 10 coins from A to B (**valid**). The new state is S1.
  - At state S1, TX1 double spends the same 10 coins from A to B (**invalid**). The state is remains S1.

LazyLedger design

- The chain is only used for **posting** arbitrary messages. Block producers don't need to care about what the messages are.
- Instead of checking transactions are **valid**, full nodes need to check they're **available**.



Data availability proofs

- You can check that a block's data is available without downloading all the data, probabilistically.
- See paper: Fraud and Data Availability Proofs. Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin.

Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam[1], Alberto Sonnino[1], and Vitalik Buterin[2]

[1] University College London
{m.albassam,a.sonnino}@cs.ucl.ac.uk
[2] Ethereum Research
vitalik@ethereum.org

Abstract. Light clients, also known as Simple Payment Verification (SPV) clients, are nodes which only download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid. Typically, instead of validating block data, they assume that the chain favoured by the blockchain's consensus algorithm only contains valid blocks, and that the majority of block producers are honest. By allowing such clients to receive fraud proofs generated by fully validating nodes that show that a block violates the protocol rules, and combining this with probabilistic sampling techniques to verify that all of the data in a block actually is available to be downloaded, we can eliminate the honest-majority assumption for block validity, and instead make much weaker assumptions about a minimum number of honest nodes that rebroadcast data. Fraud and data availability proofs are key to enabling on-chain scaling of blockchains (e.g., via sharding or bigger blocks) while maintaining a strong assurance that on-chain data is available and valid. We present, implement, and evaluate a novel fraud and data availability proof system.

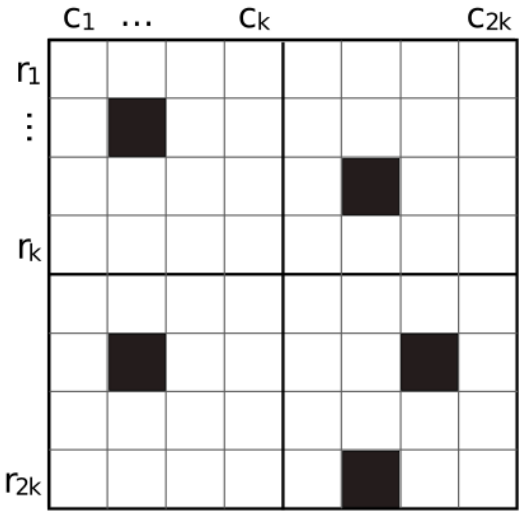
1 Introduction and Motivation

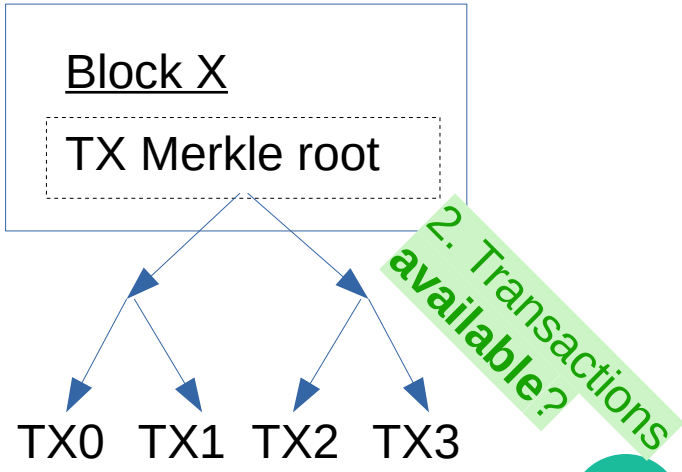
As cryptocurrencies and smart contract platforms have gained wider adoption, the scalability limitations of existing blockchains have been observed in practice. Popular services have stopped accepting Bitcoin [20] payments due to transactions fees rising as high as $20 [19,28], and Ethereum's [6] popular CryptoKitties smart contract caused the pending transactions backlog to increase six-fold [40]. Users pay higher fees as they compete to get their transactions included on the blockchain, due to on-chain space being limited, e.g., by Bitcoin's block size limit [2] or Ethereum's block gas limit [41].

While increasing on-chain capacity limits would yield higher transaction throughput, there are concerns that this would decrease decentralisation and security, because it would increase the resources required to fully download and validate the blockchain, and thus fewer users would be able to afford to run full nodes that independently validate the blockchain, requiring users to instead run

Summary of data availability proofs

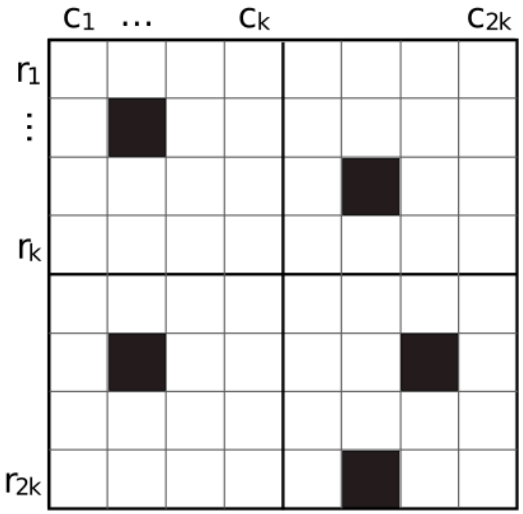
- **By erasure coding the block and splitting the block into chunks, you can be >99% sure than all of the block's data is available, by randomly sampling <1% of the block.**

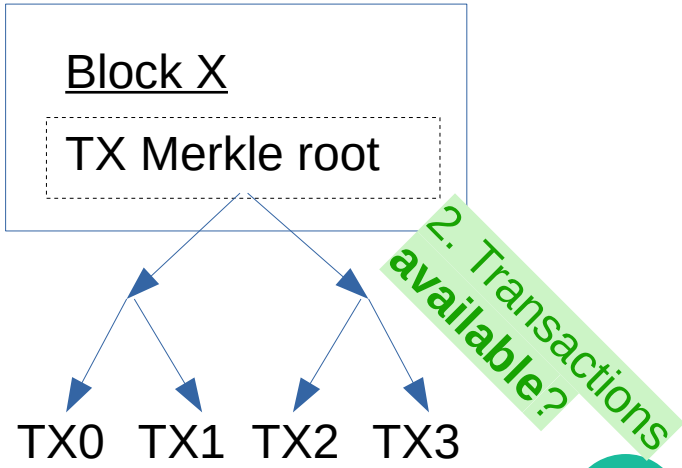




Summary of data availability proofs

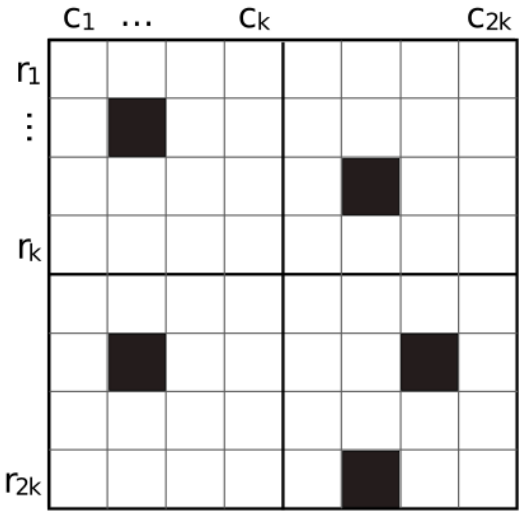
- **The number of chunks you sample is irrespective of the size of the block.**
  - So it's an ~O(1) cost to check that a block is available regardless of the size of the block.

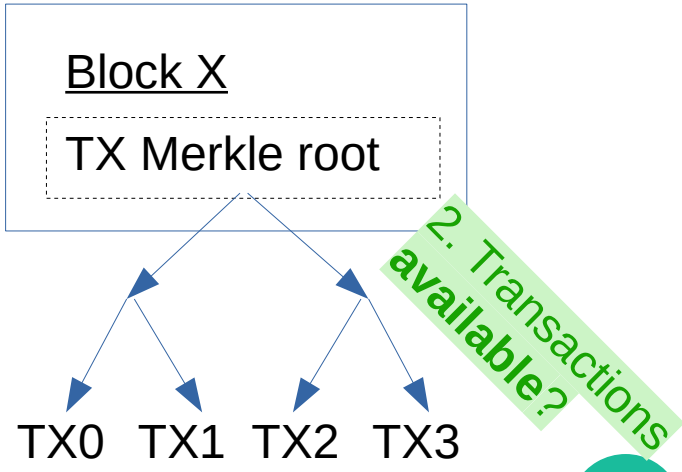




Summary of data availability proofs

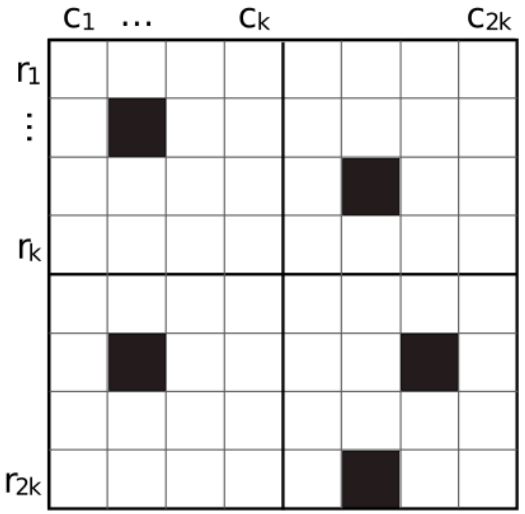
- **BUT: the bigger the block, the more nodes you need to sample chunks from the block, to be sure that the entire network has sampled the *entire* block.**

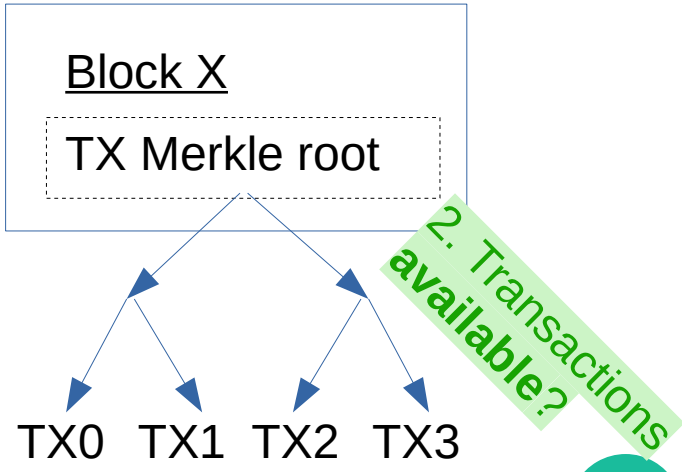




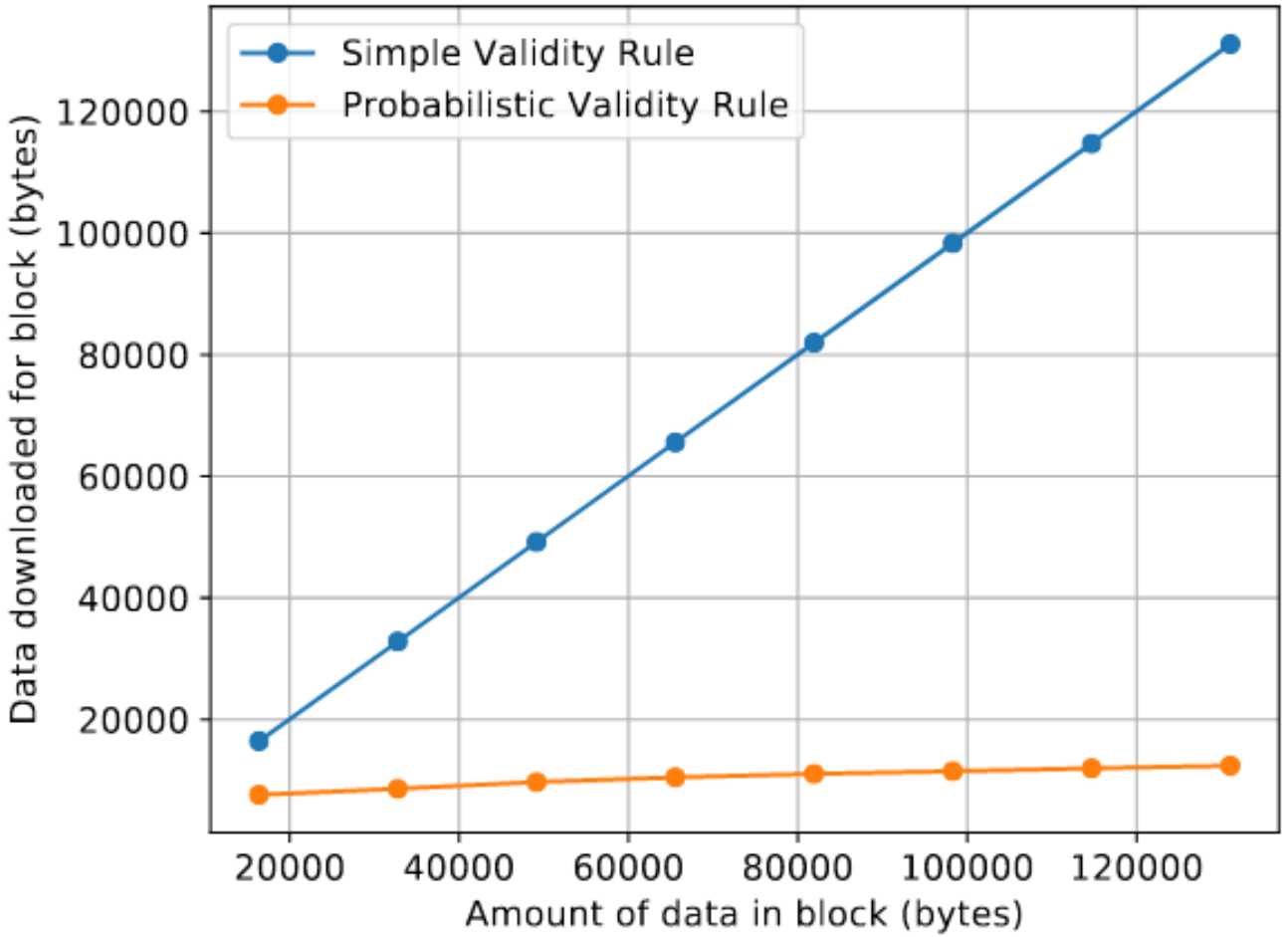
What does this mean?

- **1: You can only need to check that the block's data is available to validate it.**
  - So you can validate blocks in sub-linear time!
    Because you don't need to download the entire block.



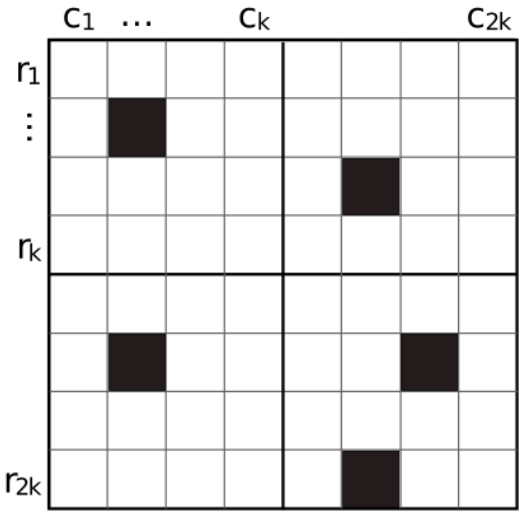


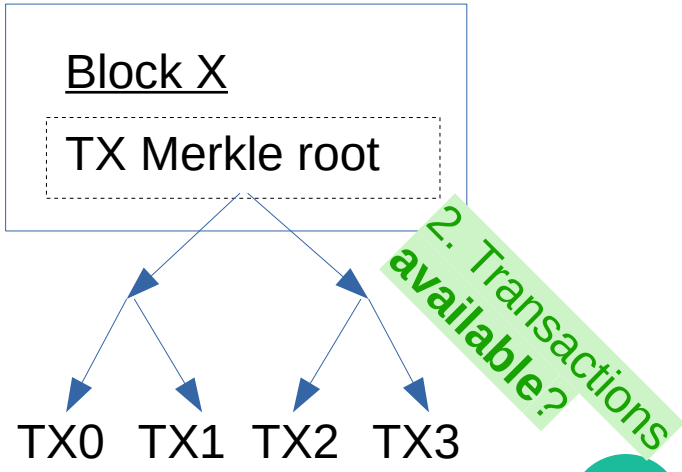
Probabilistic data availability proof size vs block size (orange line)



What does this mean?

- **2: The more nodes in the network you have, the higher the block size you can have.**
  - So the network is **scale-out** like sharding: the more nodes you have, the higher the throughput.





Why is this interesting?

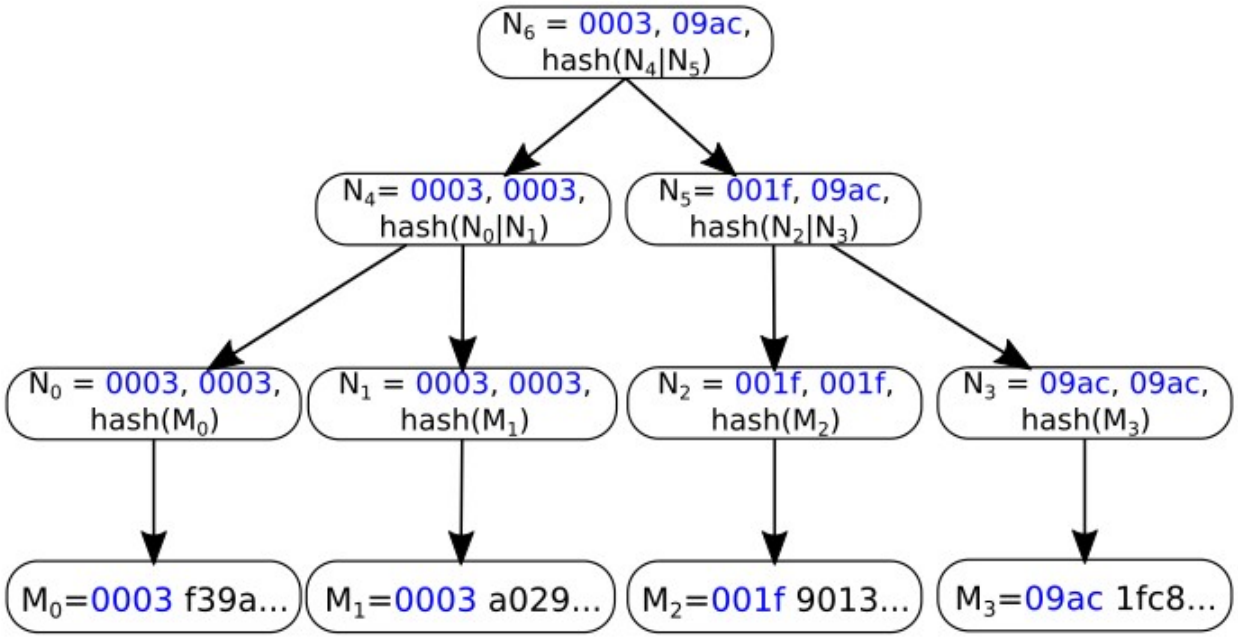
- **The world's most scalable decentralized protocol is BitTorrent.**
  - Once handled half of the Internet's traffic!
  - The more seeds you have, the more you can store and distribute.
- **By simplifying block verification to data availability verification, we can achieve similar scalability properties to BitTorrent.**

More on the execution model

- **All execution of transactions it done locally, client-side. This is cool because:**
  - No smart contract logic defined on-chain. You can write blockchain apps in any language, as they're executed locally!
  - Clients only need to download transactions they care about, not everything!
    - If the app is an optimistic rollup, clients only need to download block headers.

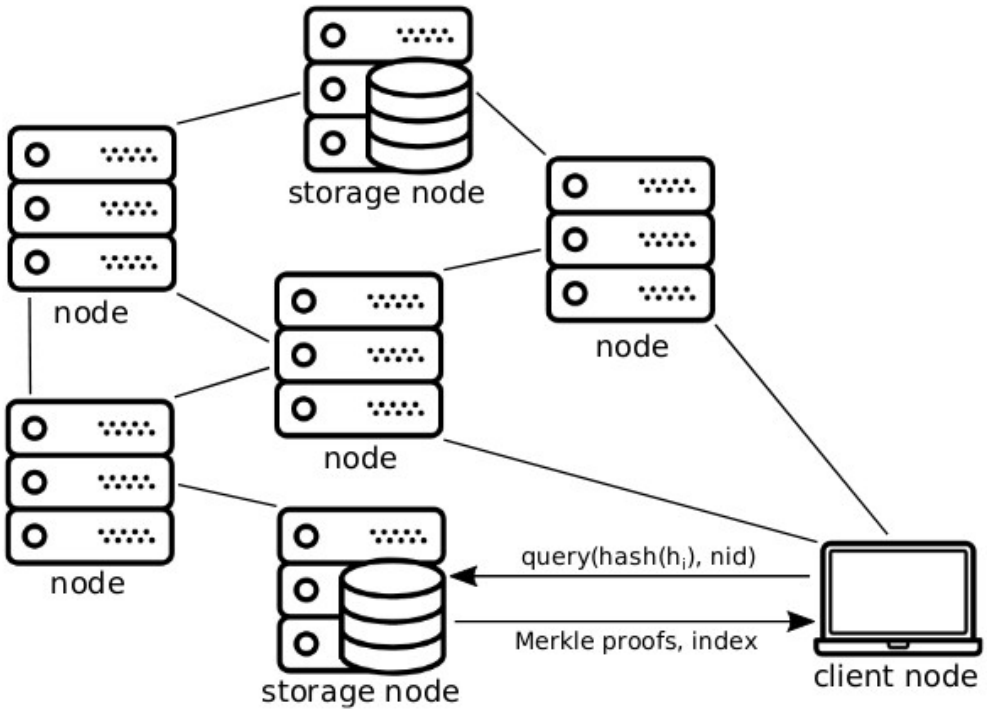
Application namespaces

- **All applications have their own ‘namespace’ that they post transactions under. The block contains a Merkle tree ordered by namespace.**



Application namespaces

- **Clients can query full storage nodes for transactions in tree relating to their application.**



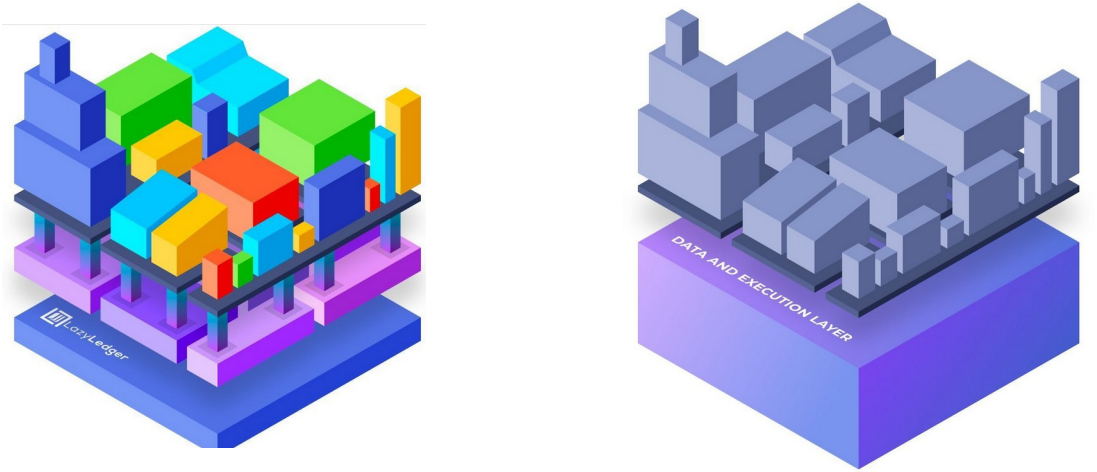
Application state sovereignty

- **On-chain applications can only directly modify the state of their own application.**
  - Because clients only download and compute state for ***their*** applications.
- **This be thought as similar to sidechains, because each application runs independently, but they all share the same chain for data availability.**
  - This is effectively heterogeneous sharding.
    - But the sharding environment is not *enshrined* into the base layer; but is opt-in.

Use case

- **Data availability layer for optimistic rollups**
  - Allows anyone for the first time in history, to deploy their own blockchain with its own execution environment, without having to deploy a new consensus layer.

LazyLedger vs Today



LazyLedger paper

- **Preprint on arXiv: https://arxiv.org/abs/1905.09274**

This document is a draft; feedback is welcome.

LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts

Mustafa Al-Bassam[1]

[1]Department of Computer Science
University College London
m.albassam@cs.ucl.ac.uk

Abstract

We propose LazyLedger, a design for distributed ledgers where the blockchain is optimised for solely ordering and guaranteeing the availability of transaction data. Responsibility for executing and validating transactions is shifted to only the clients that have an interest in specific transactions relating to blockchain applications that they use. As the core function of the consensus system of a distributed ledger is to order transactions and ensure their availability, consensus participants do not necessarily need to be concerned with the contents of those transactions. This reduces the problem of block verification to data availability verification, which can be achieved probabilistically with sub-linear complexity, without downloading the whole block. The amount of resources required to reach consensus can thus be minimised, as transaction validity rules can be decoupled from consensus rules. We also implement and evaluate several example LazyLedger applications, and validate that the workload of clients of specific applications does not significantly increase when the workload of other applications that use the same chain increase.

1 Introduction

So far, blockchain-based distributed ledger platforms such as Bitcoin [1] and Ethereum [2] have adopted similar consensus design paradigms, where the validity of the blocks proposed by block producers is determined by (*i*) whether it is the block producer's turn to propose a block and (*ii*) whether the transactions in the block are valid according to some state machine. Traditional consensus protocols such as Practical Byzantine Fault Tolerance [3] have also taken a similar approach, where consensus nodes (replicas) process transactions according to a state machine.

The scalability issues that have plagued decentralised blockchains [4] can be attributed to the fact that in order to run a node that validates the blockchain, the node must download, process and validate every transaction included in the chain. As a result, various scalability efforts have emerged including on-chain scaling via sharding [5, 6], which aims to split the state of the blockchain into multiple shards so that transactions can be processed by different consensus groups in parallel, and off-chain scaling via state channels [7, 8], which takes the approach of moving transactions off-chain and using the blockchain as a settlement layer.

However, it is also worth exploring alternative blockchain design paradigms that may be suitable for different types of applications, where nodes that need to validate the blockchain in order to determine the correct chain do not need to validate the contents of

1

How do data availability proofs work?

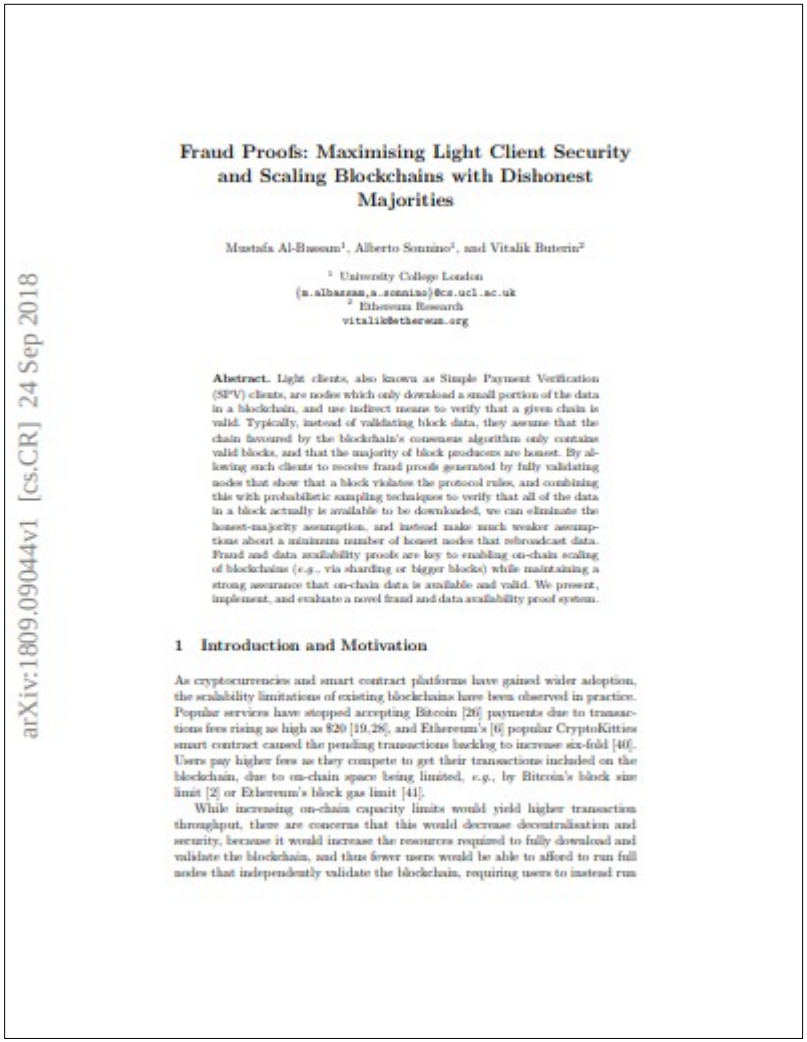
Read the full paper (33 pages)

@musalbas

- Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

  Mustafa Al-Bassam (UCL)
  Alberto Sonnino (UCL)
  Vitalik Buterin (Ethereum)

- Paper:
  - arxiv.org/abs/1809.09044
- Code:
  - github.com/musalbas/rsmt2d
  - github.com/musalbas/smt
  - github.com/asonnino/fraudproofs-prototype



Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam[1], Alberto Sonnino[1], and Vitalik Buterin[2]

[1] University College London
{m.albassam,a.sonnino}@cs.ucl.ac.uk
[2] Ethereum Research
vitalik@ethereum.org

Abstract. Light clients, also known as Simple Payment Verification (SPV) clients, are nodes which only download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid. Typically, instead of validating block data, they assume that the chain favoured by the blockchain's consensus algorithm only contains valid blocks, and that the majority of block producers are honest. By allowing such clients to receive fraud proofs generated by fully validating nodes that show that a block violates the protocol rules, and combining this with probabilistic sampling techniques to verify that all of the data in a block actually is available to be downloaded, we can eliminate the honest-majority assumption, and instead make much weaker assumptions about a minimum number of honest nodes that rebroadcast data. Fraud and data availability proofs are key to enabling on-chain scaling of blockchains (*e.g.*, via sharding or bigger blocks) while maintaining a strong assurance that on-chain data is available and valid. We present, implement, and evaluate a novel fraud and data availability proof system.

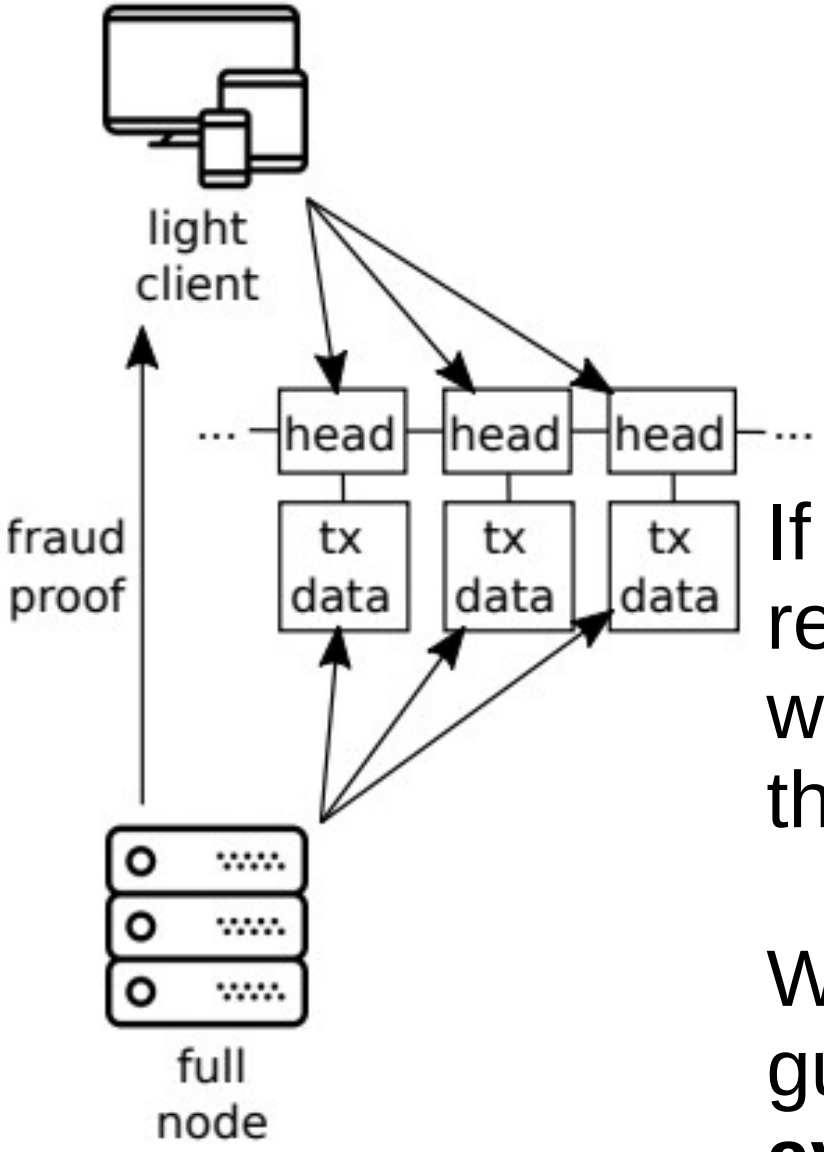
1 Introduction and Motivation

As cryptocurrencies and smart contract platforms have gained wider adoption, the scalability limitations of existing blockchains have been observed in practice. Popular services have stopped accepting Bitcoin [26] payments due to transaction fees rising as high as $20 [19,28], and Ethereum's [6] popular CryptoKitties smart contract caused the pending transactions backlog to increase six-fold [40]. Users pay higher fees as they compete to get their transactions included on the blockchain, due to on-chain space being limited, *e.g.*, by Bitcoin's block size limit [2] or Ethereum's block gas limit [41].

While increasing on-chain capacity limits would yield higher transaction throughput, there are concerns that this would decrease decentralisation and security, because it would increase the resources required to fully download and validate the blockchain, and thus fewer users would be able to afford to run full nodes that independently validate the blockchain, requiring users to instead run

The data availability problem

@musalbas

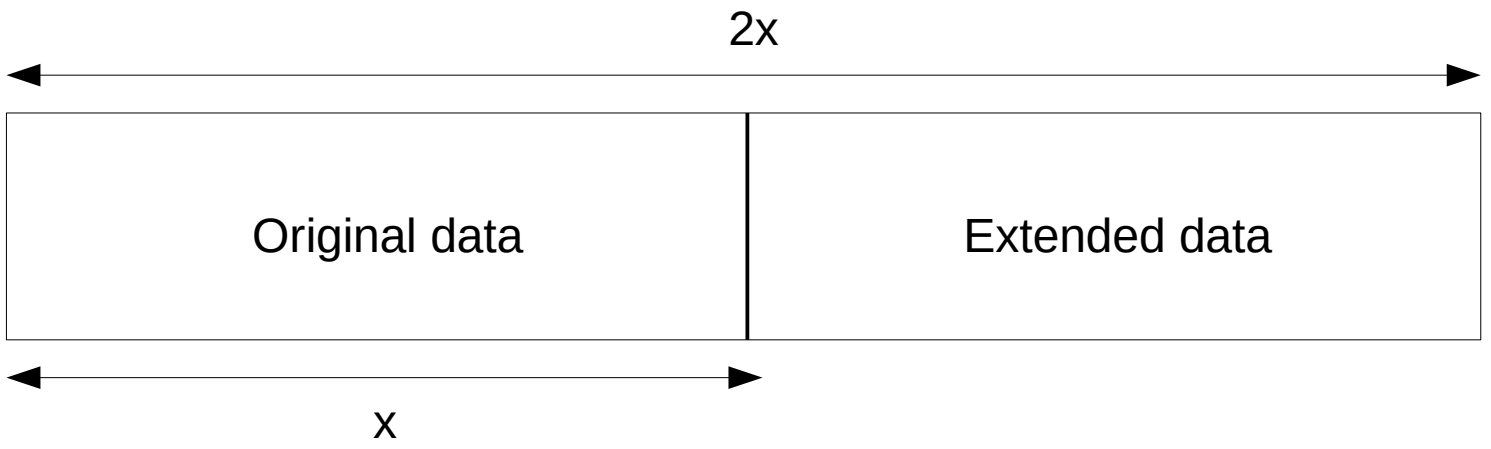


If the miner doesn't release tx data, nodes won't know the state of the chain.

We need a way to guarantee **data availability**.

Erasure coding

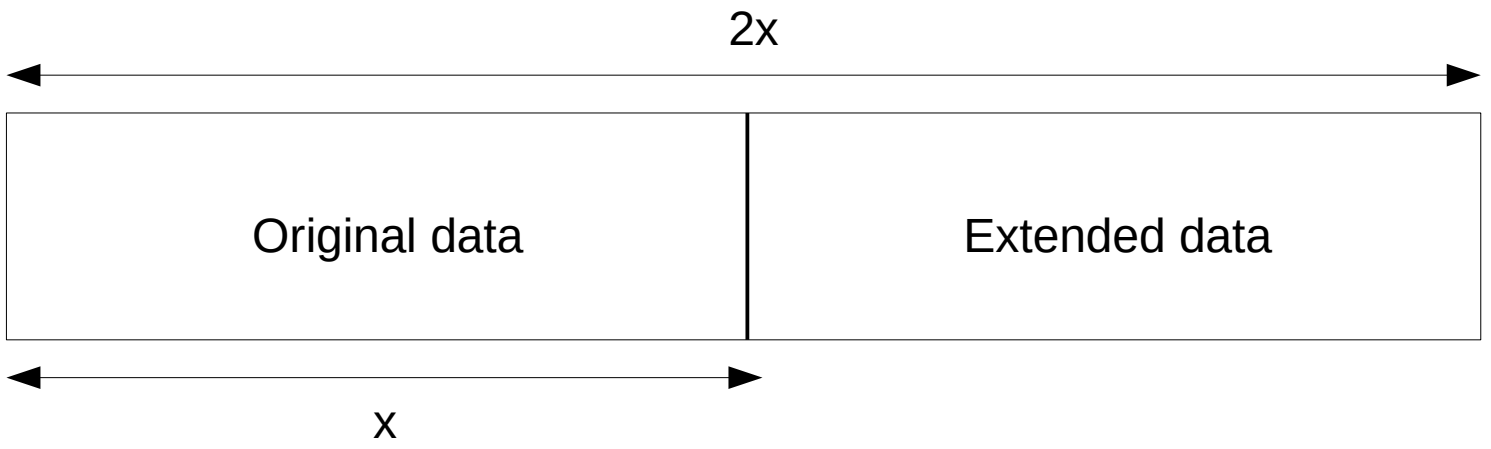
- Using erasure coding, you can extend data ***x*** pieces long to ***2x*** pieces, such that you can recover the whole data from any ***x*** pieces.



Naive data availability scheme

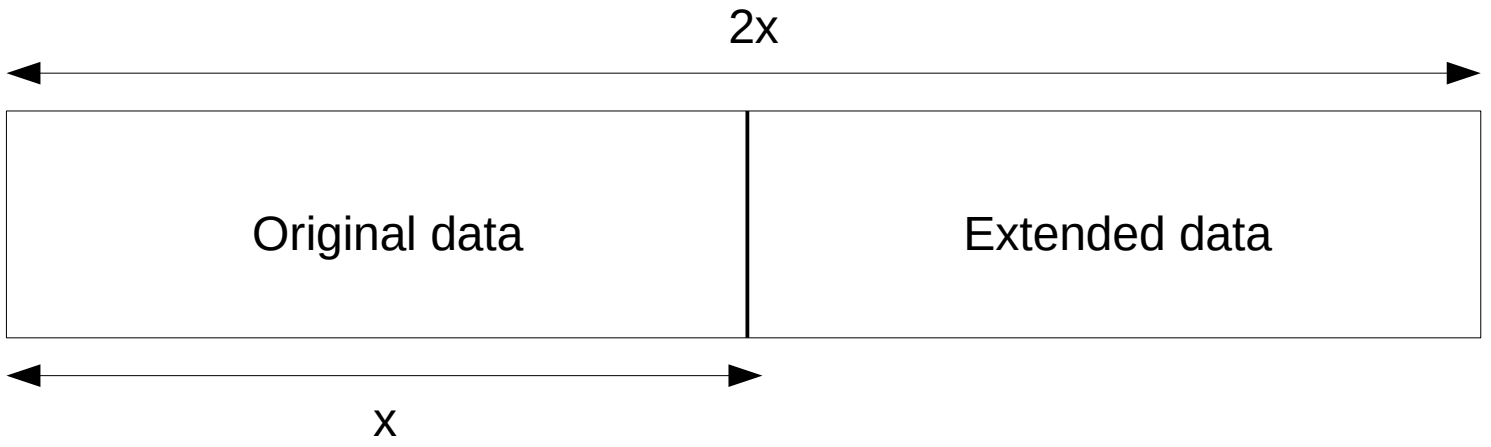
@musalbas

- We can require miners to commit to the Merkle root of the erasure coded verison of the block data. In order for a miner to hide any piece of the block, they must hide 50%.



Naive data availability scheme

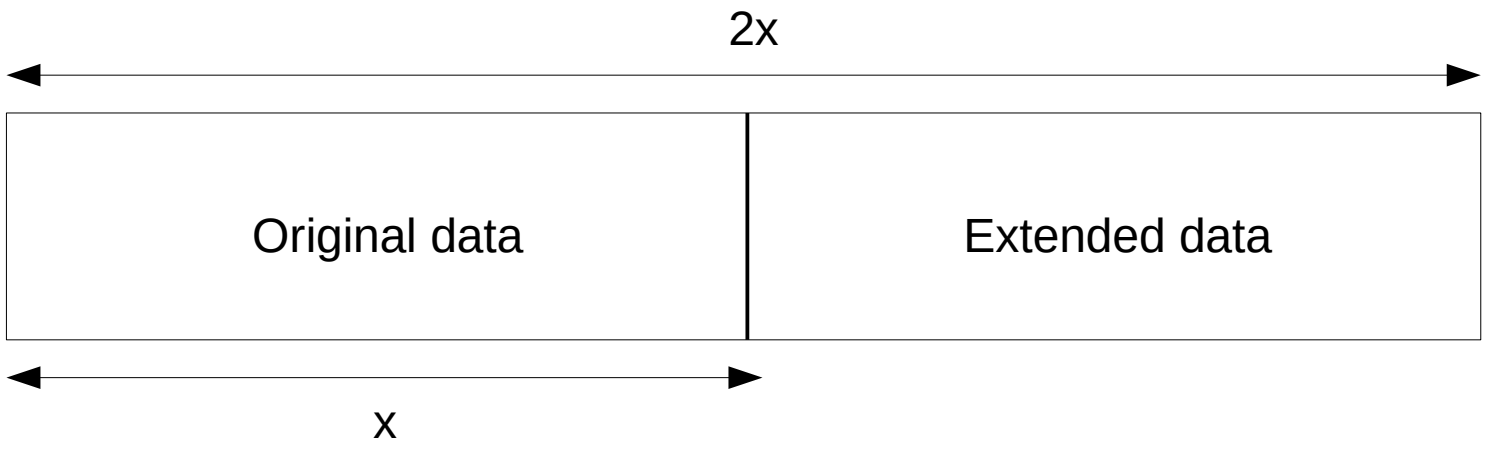
- We can require miners to commit to the Merkle root of the erasure coded verison of the block data. In order for a miner to hide any piece of the block, they must hide 50%.
- Thus clients can randomly sample parts of the block*, and if 50% is hidden, then there is a $1-2^{-s}$ chance of landing on an unavailable piece after *s* samples, in which case the block is rejected.



* with replacement

Naive data availability scheme

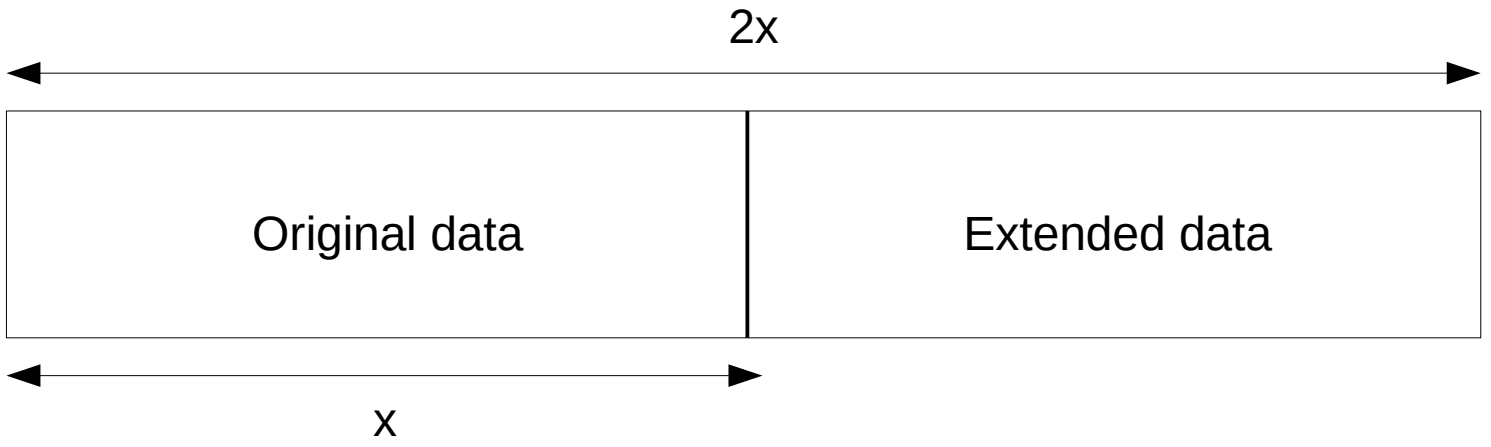
- Clients gossip pieces to full nodes, to enable full nodes to recover the data.
- There must be enough light clients making samples to reconstruct the whole data (at least **x** pieces).
  - We'll do some analysis on this in a moment.



Naive data availability scheme

@musalbas

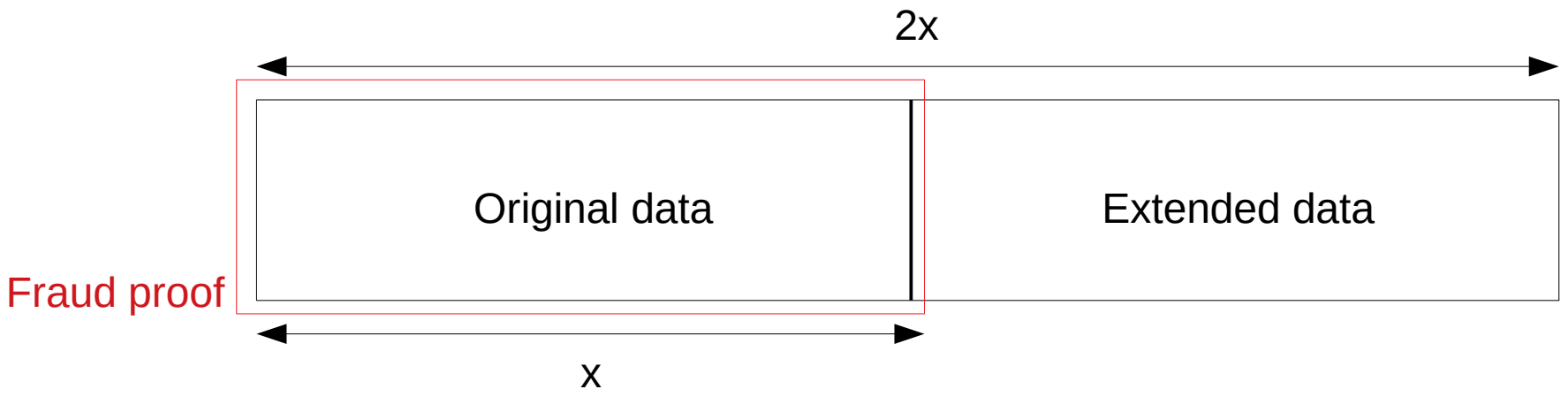
- Problem: what if the miner incorrectly generates the erasure code?



Naive data availability scheme

@musalbas

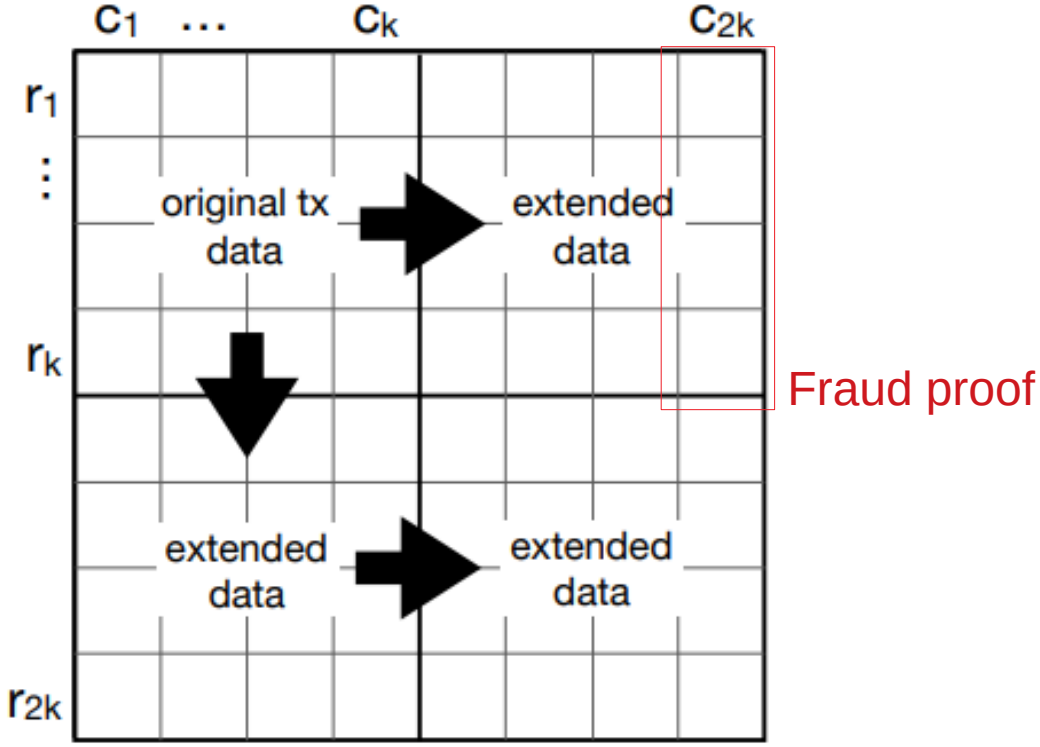
- Problem: what if the miner incorrectly generates the erasure code?
  - In order to prove this, the fraud proof consists of the entire block, as clients will need to download and regenerate the erasure code to check if it's correct.
  - That's O(blocksize). Back to square one!



Multidimensional coding

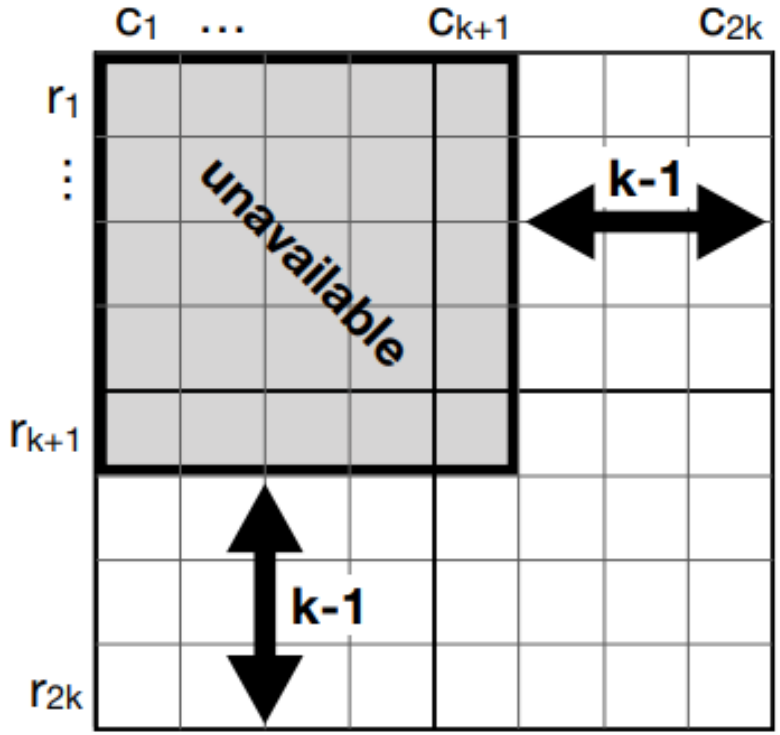
@musalbas

- We can use multidimensional coding to fix this.
- If any row or column is incorrectly generate, a fraud proof that the code is incorrectly generated is limited to that specific row or column. That's O(sqrt(blocksize)).



Multidimensional coding

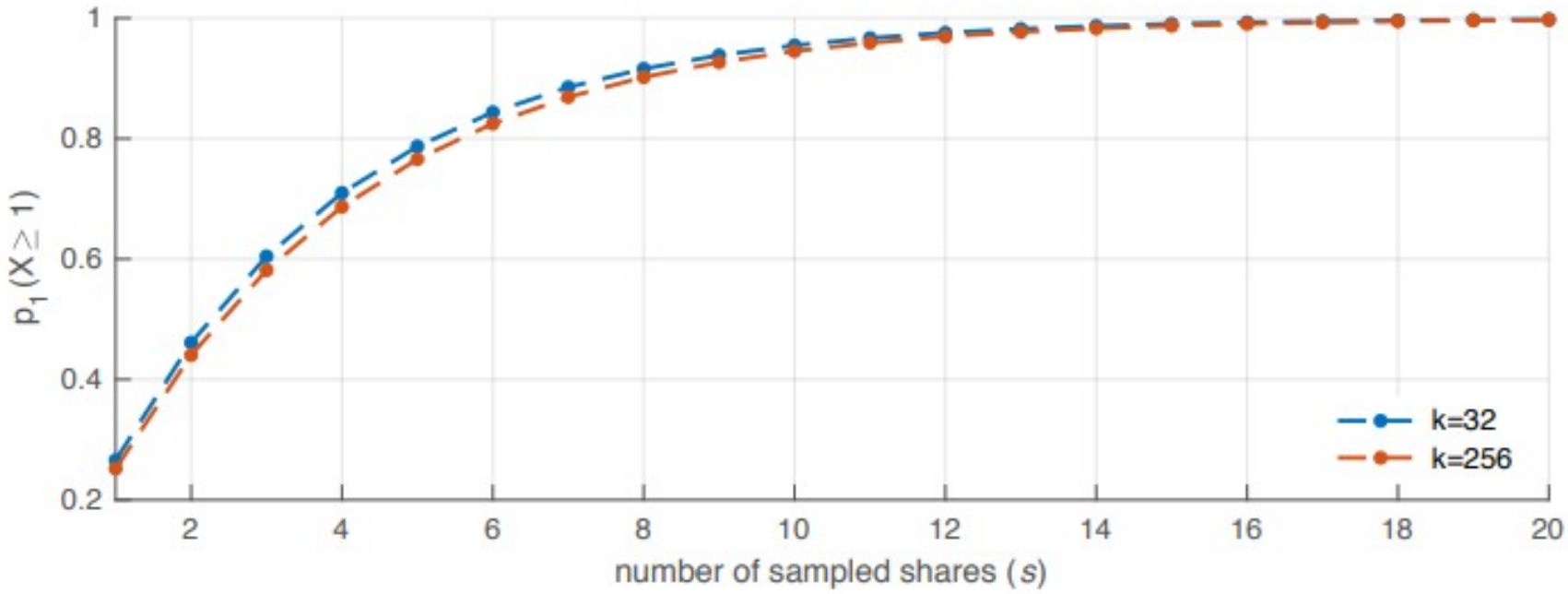
- Miner has to hide roughly 25% of the square to hide any pieces.



Probability analysis

@musalbas

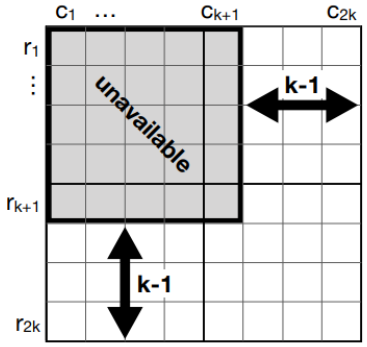
- What is the probability of a client landing on at least one available piece if the miner has hidden ~25% of the square (if sampling without replacement)?
- 60% after 3 samplings; 99% after 15 samplings.



Selective releasing of pieces

@musalbas

- What if a miner only releases pieces as clients ask for them?
  - Then the miner can always pass the sampling challenge of the first couple of hundred to thousand clients.
  - The exact number of how many clients can be fooled depends on how many samples they make each (s) and how wide is the square (k).



$p_e(Z \geq \gamma)$	$s = 2$	$s = 5$	$s = 10$	$s = 20$	$s = 50$
$k = 16$	692	277	138	69	28
$k = 32$	2805	1,122	561	280	112
$k = 64$	11,289	4,516	2,258	1,129	451
$k = 128$	>40,000	~18,000	~9,000	~4,500	1,811

Preventing selective releasing of pieces

@musalbas

- We can prevent this by assuming an enhanced network model.
  - Clients send requests anonymously.
  - The order in which requests are received by the network are uniformly random (i.e. client's sampled are interleaved). For example, using a mix net.
- This would mean that a miner would have same probability of fooling all clients, including the first ones to ask for samples.
  - Because requests are unlinkable to clients; sampling challenges cannot be satifisfied on a per-client basis.

Data availability security assumptions comparison

@musalbas

- What additional security assumptions are necessary to only accept availabile blocks?

Full nodes	Light clients	Light clients + fraud proofs
	+ 51% of consensus is honest + synchronous gossiping network	+ minimum number of light clients (few hundred) + synchronous gossiping network

Performance: space/bandwidth

@musalbas

Object	Space complexity	250KB block	1MB block
State fraud proof	$O(p + p\log(d) + w\log(s) + w)$	14,090b	14,410b
Availability fraud proof	$O(d^{0.5} + d^{0.5}\log(d^{0.5}))$	5,120b	12,288b
Single sample response	$O(shareSize + \log(d))$	320b	368b
Header	$O(1)$	128b	128b
Header + axis roots	$O(d^{0.5})$	2,176b	4,224b

Table 2: Worst case space complexity and illustrative sizes for various objects for 250KB and 1MB blocks. p represents the number of transactions in a period, w represents the number of witnesses for those transactions, d is short for dataLength, and s is the number of key-value pairs in the state tree. For the illustrative sizes, we assume that a period consists of 10 transactions, the average transaction size is 225 bytes, and that conservatively there are 2^{30} non-default nodes in the state tree.

Performance: computation

@musalbas

Action	Time complexity	250KB block	1MB block
[G] State fraud proof	$O(b + p\log(d) + w)$	289.78 ms	981.88 ms
[V] State fraud proof	$O(p + p\log(d) + w)$	1.50 ms	1.50 ms
[G] Availability fraud proof	$O(d^2 + d^{0.5}\log(d^{0.5}))$	7.96ms	50.88ms
[V] Availability fraud proof	$O(d + d^{0.5}\log(d^{0.5}))$	0.05ms	0.19ms
[G] Single sample response	$O(log(d^{0.5}))$	< 0.00001ms	< 0.00001ms
[V] Single sample response	$O(log(d^{0.5}))$	< 0.00001ms	< 0.00001ms

Table 3: Worst case time complexity and benchmarks for various actions for 250KB and 1MB blocks (mean over 10 repeats), where [G] means generate and [V] means verify. p represents the number of transactions in a period, b represents the number of transactions in the block, w represents the number of witnesses for those transactions, d is short for dataLength, and s is the number of key-value pairs in the state tree. For the benchmarks, we assume that a period consists of 10 transactions, the average transaction size is 225 bytes, and each transaction writes to one key in the state tree.

Questions?