Dietcoin: slimming Bitcoin for your smartphone

Davide Frey\textsuperscript{1}, Marc X. Makkes\textsuperscript{2}, Pierre-Louis Roman\textsuperscript{1}, François Taïani\textsuperscript{1}, Spyros Voulgaris\textsuperscript{2}

\textsuperscript{1} Univ Rennes, Inria, CNRS, IRISA
\textsuperscript{2} Vrije Universiteit Amsterdam
Bitcoin

Be your own bank

Payment system

- Trustless
- Distributed
- Attacker-proof?
- Scalable?

Made possible thanks to its blockchain
The Bitcoin blockchain

Blockchain
- Ledger of transactions
- Tamper-proof
- **Verifiable**

Bitcoin blockchain
- Open membership
- Proof-of-Work consensus
  - Byzantine-tolerant
  - 1 block / 10 minutes

Two operations/actors
- append(block) by miners
- read(chain) by full/light nodes
The Bitcoin blockchain

Blockchain
- Ledger of transactions
- Tamper-proof
- **Verifiable**

Bitcoin blockchain
- Open membership
- Proof-of-Work consensus
  - Byzantine-tolerant
  - 1 block / 10 minutes

Two operations/actors
- `append(block)` by miners
- `read(chain)` by full/light nodes
Blockchain data structure

Block

- Encapsulates transactions
- Immutable, provides safety
- Difficult to create (work)

Blockchain

- Chain with the most work
- Probabilistic consistency

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4 B</td>
</tr>
<tr>
<td>Prev block</td>
<td>32 B</td>
</tr>
<tr>
<td>Tx hash</td>
<td>32 B</td>
</tr>
<tr>
<td>Timestamp</td>
<td>4 B</td>
</tr>
<tr>
<td>Difficulty</td>
<td>4 B</td>
</tr>
<tr>
<td>Nonce</td>
<td>4 B</td>
</tr>
</tbody>
</table>

[ tx_1, tx_2, ..., tx_n ]
Blockchain data structure

Block
- Encapsulates transactions
- Immutable, provides safety
- Difficult to create (work)

Blockchain
- Chain with the most work
- Probabilistic consistency
Making a transaction
Making a transaction
Making a transaction
Making a transaction
Making a transaction

Tony

Lenny

baadc0de

Block

[...]

baadc0de

[...]
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
Time and bandwidth expensive bootstrap
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
- Time and bandwidth expensive bootstrap

<table>
<thead>
<tr>
<th>← deadc0de header</th>
<th>← deadbeef header</th>
<th>← 8badf00d header</th>
</tr>
</thead>
<tbody>
<tr>
<td>transactions</td>
<td>transactions</td>
<td>transactions</td>
</tr>
</tbody>
</table>

100 GiB
Full node
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
- Time and bandwidth expensive bootstrap

```
← deadc0de
header
transactions
← deadbeef
header
transactions
← 8badf00d
header
transactions
```

34 MiB
Light node

100 GiB
Full node
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
- Time and bandwidth expensive bootstrap

What if a light node receives blocks with fake transactions?
Verifying transactions

Money can only be spent once

Diagram showing transactions with inputs and outputs.
Verifying transactions

Money can only be spent once

Diagram showing transactions with inputs and outputs.
Verifying transactions

Money can only be spent once

Diagram showing transactions with in and out labels.
Verifying transactions

Money can only be spent once

\[
\begin{array}{c}
\text{in} & \text{out} \\
1 & 2 \\
3 & 4 \\
5 & \\
6 & \\
\end{array}
\]
Verifying transactions

Money can only be spent once

```
in  out
1  2
3  4
5  5
6
```
Verifying transactions

Money can only be spent once

Diagram showing transactions with in and out indicators.
Verifying transactions

Money can only be spent once

Diagram showing transactions with inputs and outputs.
Verifying transactions

Money can only be spent once

UTXO set

Set of Unspent Transaction Outputs (U TX O)
Rapidly growing UTXO set

Blockchain size (GiB)
UTxO set size (GiB)

Blockchain
UTxO set
Problem statement

- Light nodes don’t verify transactions
- Building the UTXO requires a lot of bandwidth and time
- (and it’s getting worse)
Intuition

Make the UTXO queriable by light nodes

Pros

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

Cons

- Diet nodes consume more bandwidth than light nodes
Intuition

Make the UTXO queriable by light nodes

Pros
- Diet node = light node + transaction verification
- Fast bootstrap, improved security

Cons
- Diet nodes consume more bandwidth than light nodes
Intuition

Make the UTXO queriable by light nodes

Pros

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

Cons

- Diet nodes consume more bandwidth than light nodes

What if a diet node receives a fake UTXO set?
Hash of UTXO set

Pros
- It works!

Cons
- Costly for small queries
Hash of UTXO set

Pros
- It works!

Cons
- Costly for small queries
Merkle tree: efficient proof of inclusion

\[ H_{ABCDEFGH} \]

\[ H_{ABCD} \]
\[ H_{AB} \]
\[ H_A \]
\[ A \]

\[ H_B \]
\[ B \]

\[ H_C \]
\[ C \]

\[ H_D \]
\[ D \]

\[ H_{EF} \]
\[ H_E \]
\[ E \]

\[ H_F \]
\[ F \]

\[ H_{GH} \]
\[ H_G \]
\[ G \]

\[ H_H \]
\[ H \]

Download
Reconstruct
Merkle tree: efficient proof of inclusion

\[ \text{H}_{\text{ABCDEF GH}} \]

\[ \text{H}_{\text{ABCD}} \]

\[ \text{H}_{\text{AB}} \quad \text{H}_{\text{CD}} \]

\[ \text{H}_{\text{A}} \quad \text{H}_{\text{B}} \quad \text{H}_{\text{C}} \quad \text{H}_{\text{D}} \]

\[ \text{A} \quad \text{B} \quad \text{C} \quad \text{D} \]

Download
Reconstruct
Merkle tree: efficient proof of inclusion

- $H_{ABCD}$
  - $H_{AB}$
    - $H_A$
      - A
    - $H_B$
      - B
  - $H_{CD}$
    - $H_C$
      - C
    - $H_D$
      - D
  - $H_{EF}$
    - $H_E$
      - E
    - $H_F$
      - F
  - $H_{GH}$
    - $H_G$
      - G
    - $H_H$
      - H

- $H_{ABCDEFGH}$
  - $H_{ABCD}$
  - $H_{EF}$
  - $H_{GH}$

Download
Reconstruct
Merkle tree: efficient proof of inclusion
Merkle tree: efficient proof of inclusion
Merkle tree: efficient proof of inclusion

\[ H_{\text{ABCDEFGH}} \]

\[ H_{\text{ABCDEF}} \]

\[ H_{\text{EF}} \]

\[ H_{\text{GH}} \]

\[ H_{\text{AB}} \]

\[ H_{\text{ABCD}} \]

\[ H_{\text{AB}} \]

\[ H_{\text{E}} \]

\[ H_{\text{F}} \]

\[ H_{\text{G}} \]

\[ H_{\text{H}} \]
Merkle tree of UTXO set

Pros
- Diet node bandwidth consumption: 1.5 MiB of query per block

Cons
- Full node storage overhead: 915 MiB
- Lots of leaf insertions and deletions
Merkle tree of UTXO set

### Pros
- Diet node bandwidth consumption: 1.5 MiB of query per block

### Cons
- Full node storage overhead: 915 MiB
- Lots of leaf insertions and deletions
Merkle tree of sharded UTXO set

Pros

- Diet node bandwidth consumption: 4.15 MiB of query per block
- Full node storage overhead: 64 MiB
- Parameterized trade-off $k$: bandwidth consumption vs storage overhead
- Stable tree: no insertion, no deletion
Merkle tree of sharded UTXO set

Pros

- Diet node bandwidth consumption: 4.15 MiB of query per block
- Full node storage overhead: 64 MiB
- Parameterized trade-off $k$: bandwidth consumption vs storage overhead
- Stable tree: no insertion, no deletion
### Roughly estimated performances

<table>
<thead>
<tr>
<th></th>
<th>Diet node bandwidth</th>
<th>Full node storage overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full UTXO</strong></td>
<td>1.5 GiB</td>
<td>0</td>
</tr>
<tr>
<td><strong>Merkle tree UTXO</strong></td>
<td>1.5 MiB</td>
<td>915 MiB</td>
</tr>
<tr>
<td><strong>Sharded UTXO</strong></td>
<td>4.15 MiB</td>
<td>64 MiB</td>
</tr>
</tbody>
</table>
Evaluation schema

Diet node
- Register
- Forward headers & blocks
- Query UTXOs
- Reply UTXOs

Full node
- Blocks
- UTXO set
- UTXO Merkle tree

Metrics
- Diet node bandwidth
- Full node storage overhead
- Phone battery consumption
- Time overhead

Baselines and scenarios
- Full node
- Light node
- Diet node with full UTXO
- Diet node with sharded UTXO
Evaluation schema

Diet node
- Register
- Query UTXOs
- Reply UTXOs
- Forward headers & blocks
- Headers

Dietcoin replayer
- Register
- Query all headers
- Send all headers
- Query blocks one by one
- Send blocks
- Headers
- UTXO set
- UTXO Merkle tree

Bitcoin full node
- Register
- Query all headers
- Send all headers
- Blocks
- UTXO set

Metrics
- Diet node bandwidth
- Full node storage overhead
- Phone battery consumption
- Time overhead

Baselines and scenarios
- Full node
- Light node
- Diet node with full UTXO
- Diet node with sharded UTXO
Evaluation schema

**Metrics**
- Diet node bandwidth
- Full node storage overhead
- Phone battery consumption
- Time overhead

**Baselines and scenarios**
- Full node
- Light node
- Diet node with full UTXO
- Diet node with sharded UTXO
Recap of Dietcoin

The good
- Shortcut the full verification process
- Decoupled storage

The bad
- Inherent overhead for full nodes
- Non-optimal bandwidth consumption

The future
- Evaluation
- Decoupled storage $\Rightarrow$ DHT
- Add pubKey bloom filter in headers
UTxO - Merkle tree

Number of items in the UTxO = 15 M
Number of nodes in the tree = 2 \times 15 M = 30 M
Server storage overhead: 30 M \times 32 B \approx 915 \text{MiB}

Number of hashes to reconstruct the merkle root: \log_2(30 M) - 1 \approx 25

Typical block verification:
- 2000 inputs minimum
- Needs 2000 proof of existence in the tree
- Client bandwidth consumption: 2000 \times 25 \times 32 B \approx 1.5 \text{MiB}

Balancing the tree, storage overhead on servers
UTxO - Sharding with Merkle tree

Sharding policy: use first \( k \) bits, e.g., \( k = 20 \)
Number of shards = \( 2^{20} \)
Number of nodes in the tree = \( 2 \times 2^{20} - 1 \approx 2^{21} \)
Server storage overhead: \( 2^{21} \times 32 \text{B} = 64 \text{MiB} \)

UTxO set size = 1.5 GiB
Shard size = 1.5 GiB \( \div 2^{20} \approx 1.5 \text{KiB} \)
Number of hashes to reconstruct the merkle root: \( \log_2(2^{21}) - 1 = 20 \)

Typical block verification:

- \( \sim 2000 \) inputs
- With random sharding, needs 2000 shards
- Client bandwidth consumption: \( 2000 \times (1.5 \text{KiB} + 20 \times 32 \text{B}) \approx 4.15 \text{MiB} \)

Parameterized trade-off bandwidth vs storage overhead