Shortcutting the Bitcoin verification process for your smartphone

Pierre-Louis Roman

University of Rennes
Hi

PhD: Exploring heterogeneity in loosely consistent decentralized data replication
Defense on 2018-12-18 in Rennes, France
  Supervisors: François Taïani and Davide Frey

Work on decentralized systems, try to work on the theory of distributed algorithms.

---

**Dietcoin**\(^a\): shortcutting the Bitcoin verification process for your smartphone.
Davide Frey, Marc X. Makkes, Pierre-Louis Roman, François Taïani, Spyros Voulgaris.

\(^a\)Not Dietbitcoin
The Bitcoin payment system

- Public
- **Trustless**
- Decentralized
- Secured?
- Scalable?

"Be your own bank"

Made possible thanks to its blockchain
Distributed ledger\(^1\) and blockchain

Distributed ledger object
- Append-only queue of records
- Two operations: append(), read()
- Distributed $\Rightarrow$ Consensus

Blockchain
- Mostly for decentralized ledgers
- Chaining prevents modification
- Commits with blocks

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- Trustless: fully verifiable
- Records financial transactions
- append(block) by miners

\(^1\)More in Chapter 16.7 of Michel Raynal’s 2018 book
Distributed ledger\(^1\) and blockchain

Distributed ledger object
- Append-only queue of records
- Two operations: append(), read()
- Distributed $\Rightarrow$ Consensus

Blockchain
- Mostly for decentralized ledgers
- Chaining prevents modification
- Commits with blocks

---

\(^1\)More in Chapter 16.7 of Michel Raynal’s 2018 book

---

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- Trustless: fully verifiable
- Records financial transactions
- append(block) by miners
Distributed ledger\(^1\) and blockchain

Distributed ledger object
- Append-only queue of records
- Two operations: `append()`, `read()`
- Distributed $\implies$ Consensus

Blockchain
- Mostly for decentralized ledgers
- Chaining prevents modification
- Commits with blocks

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- Trustless: fully verifiable
- Records financial transactions
- `append(block)` by miners

---

\(^1\)More in Chapter 16.7 of Michel Raynal’s 2018 book
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Valid transactions

- Spend existing money
- Spend available money
- Do not create money
- Are signed by the owner

Nodes store the set of Unspent Transaction Outputs (UTXO set)
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 1, 2, 3 \}
Verifying blocks

Update the UTXO set on valid blocks

\[
\text{UTXO set} = \{ 4, 2, 3, 4, 5 \}
\]
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 2, 3, 4, 5 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 2, 3, 4, 5, 6, 7, 8 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 6, 7, 8 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 6, 7, 8 \}
Verification process

- Replicate chain locally
- Verify headers then transactions
- Verify based on agreed upon rules

Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Verification process

- Replicate chain locally
- Verify headers then transactions
- Verify based on agreed upon rules

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Verification process

- Replicate chain locally
- Verify headers then transactions
- Verify based on agreed upon rules

Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Rapidly growing chain and UTXO set
Scalability and security problem

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

What if a light node receives a block with incorrect transactions?
Intuition

Make the UTXO set queriable by light nodes

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

- Diet nodes consume more bandwidth than light nodes
Intuition

Make the UTXO set queriable by light nodes

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

- Diet nodes consume more bandwidth than light nodes
**Intuition**

Make the UTXO set queriable by light nodes

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

- Diet nodes consume more bandwidth than light nodes

**WHAT IF**

A diet node receives incorrect UTXOs?
New rule: full nodes verify $H_{\text{UTXO}}$ upon block reception.

It works!

A light node has to download the entire UTXO set, even for small queries.
New rule: full nodes verify $H_{\text{UTXO}}$ upon block reception

It works!

A light node has to download the entire UTXO set, even for small queries
Merkle tree of UTXO set (2012 post from Andrew Miller)

- Precise queries, no superfluous UTXOs sent
- Big tree, full node storage overhead: 3 GiB
  - $|\text{UTXOs}| = 50 \text{ M} \Rightarrow |\text{hashes}| = 100 \text{ M} \Rightarrow \text{tree size} = 3 \text{ GiB}$
- Lots of leaf insertions and deletions, needs self-balancing tree?
Merkle tree of UTXO set (2012 post from Andrew Miller)

Precise queries, no superfluous UTXOs sent

Big tree, full node storage overhead: 3 GiB

\[ |UTXOs| = 50 \text{ M} \implies |hashes| = 100 \text{ M} \implies \text{tree size} = 3 \text{ GiB} \]

Lots of leaf insertions and deletions, needs self-balancing tree?
Merkle tree of sharded UTXO set (Dietcoin)

- Full node storage overhead: 256 MiB
  \[ M = 1 \text{ KiB} \Rightarrow k = 22 \Rightarrow |\text{hashes}| = 2^{23} \]
- Diet node bandwidth consumption: 13 MiB of query per block
  \[ 10000 \text{ shards} \times (0.64 \text{ KiB} + 22 \times 32 \text{ B}) = 13 \text{ MiB} \]
- Parameterized trade-off $k$: bandwidth consumption vs storage overhead
- Predictable tree: no insertion, no deletion $\Rightarrow$ Subchain verification
Merkle tree of sharded UTXO set (Dietcoin)

- Full node storage overhead: 256 MiB
  \[ M = 1 \text{KiB} \Rightarrow k = 22 \Rightarrow |\text{hashes}| = 2^{23} \]
- Diet node bandwidth consumption: 13 MiB of query per block
  \[ 10000 \text{ shards} \times (0.64 \text{KiB} + 22 \times 32 \text{B}) = 13 \text{MiB} \]
- Parameterized trade-off \( k \): bandwidth consumption vs storage overhead
- Predictable tree: no insertion, no deletion \( \Rightarrow \) Subchain verification
Codependency blocks and UTXO set

Block $k-1$
- Header
- MRoot UTXO $k-1$
- Transactions

Block $k$
- Header
- MRoot UTXO $k$
- Transactions

MRoot UTXO $k-1$ Verifies MRoot UTXO $k$

Shard $A$, Shard $B$, Shard $C$, Shard $D$
Subchain verification

- Trust a block, verify all the next ones
- Shift the trust from the genesis block to any block

Effectively shortcuts the verification process
Apply sharding policy
Block_{k-2}

Header

\[
\text{MRoot UTXO}_{k-2}
\]

Transactions


Block_{k-1}

Header

\[
\text{MRoot UTXO}_{k-1}
\]

\[ \rightarrow A \]

\[ A \rightarrow B \]


Block_{k}

Header

\[
\text{MRoot UTXO}_{k}
\]

\[ \rightarrow 6 \]

\[ 4, 5 \rightarrow 7 \]


Shard_A \text{ ? } Shard_B \text{ ?}

Partial MTree ?
Block_{k-2}

- Header
- MRoot UTXO_{k-2}
- Transactions

Block_{k-1}

- Header
- MRoot UTXO_{k-1}
  - A
  - A → B

Block_k

- Header
- MRoot UTXO_k
  - 6
  - 4, 5 → 7

Shard_A Shard_B

Pierre-Louis Roman (Univ Rennes)
ShardB ShardA
Header
→ 6
4, 5 → 7
Block
k-1

MRoot UTXO
k-1
Transactions

→ A
A → B

Block
k

Header

MRoot UTXO
k

Check

→ A
MRoot UTXO
k-2

Trusted

ShardA ShardB

Pierre-Louis Roman (Univ Rennes) Dietcoin 2018-11-18 — Master Workshop Layer 1 16 / 20
Shard A Shard B
Header
6
Block
4, 5 → 7
Block
MRoot UTXO
k-1
MRoot UTXO
k
MRoot UTXO
k-1
→ A
A → B
Header
Block
MRoot UTXO
k-2
Transactions
Trusted
Check
MRoot UTXO
k-1
MRoot UTXO
k
Transactions
Trusted
Check
Pierre-Louis Roman (Univ Rennes) Dietcoin 2018-11-18 — Master Workshop Layer 1
Shard

Shard

Header

Block

→

6

4, 5 → 7

Block

→

A

A → B

Header

Transactions

MRoot UTXO

MRoot UTXO

MRoot UTXO

MRoot UTXO

Pierre-Louis Roman (Univ Rennes)

Dietcoin

2018-11-18 — Master Workshop Layer 1
Apply sharding policy

Block$_{k-2}$

Header

MRoot UTXO$_{k-2}$

Transactions

Trusted

Block$_{k-1}$

Header

MRoot UTXO$_{k-1}$

→ A

A → B

Block$_k$

Header

MRoot UTXO$_k$

→ A

B, C → A

Apply sharding policy

MRoot UTXO$_{k-1}$

Shard$_A$

Shard$_B$
Block_{k-2}:
- Header
- MRoot UTXO_{k-2}
- Transactions

Trust

Block_{k-1}:
- Header
- MRoot UTXO_{k-1}
- A → A
- A → B

Block_{k}:
- Header
- MRoot UTXO_{k}
- A → A
- B, C → A

Shard_A, Shard_B, Shard_C
Shard A → B

MRoot UTXO_{k-1} → A

A → B

MRoot UTXO_{k} → A

B, C → A

Check

MRoot UTXO_{k-1}

\(\text{Shard}_A\) \(\text{Shard}_B\) \(\text{Shard}_C\)
ShardB → ShardA

Block_{k-2}

Header

MRoot UTXO_{k-2}

Transactions

Trusted

Block_{k-1}

Header

MRoot UTXO_{k-1}

→ A

→ A → B

Block_k

Header

MRoot UTXO_k

→ A

B, C → A

---

ShardA | ShardB | ShardC

---

Pierre-Louis Roman (Univ Rennes) Dietcoin 2018-11-18 — Master Workshop Layer 1
Block\textsubscript{k-2}:
- Header
- MRoot UTXO\textsubscript{k-2}
- Transactions

Block\textsubscript{k-1}:
- Header
- MRoot UTXO\textsubscript{k-1}
- A → A
- A → B

Block\textsubscript{k}:
- Header
- MRoot UTXO\textsubscript{k}
- B, C → A

Shard\textsubscript{A}, Shard\textsubscript{B}, Shard\textsubscript{C}
Adaptive number of UTXO shards

- Global parameter $M$: cap on the average shard size (e.g., 1 KiB)
- New rule: split all shards in two when average shard size above $M$
- Adds a layer to the UTXO Merkle tree

Updating all the tree prevents subchain verification (happen rarely)
Evaluation goal

Metrics

- Bandwidth consumption
- Storage consumption
- Light node battery consumption

Evaluation setup

- Use a historical dataset
  - Replay the Bitcoin chain
- Use popular code
  - Extend the bitcoinj library with Dietcoin features
- Light node in a realistic environment
  - Hide the complexity in the server
Evaluation goal

Metrics
- Bandwidth consumption
- Storage consumption
- Light node battery consumption

Evaluation setup
- Use a historical dataset
  - Replay the Bitcoin chain
- Use popular code
  - Extend the bitcoinj library with Dietcoin features
- Light node in a realistic environment
  - Hide the complexity in the server
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test other protocol improvements
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test other protocol improvements
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test other protocol improvements
Summary

- Diet nodes can verify the correctness of subchains of blocks
- Diet nodes shortcut the verification process

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

Future work

- Evaluation
- Decoupled storage $\Rightarrow$ DHT
- Other sharding policies
- Combine with Non-interactive Proofs of Proof-of-Work (NiPoPoWs)?
- Combine with Ethereum Casper?