Hello, SBC!
I’m Boyma.

(@boymanjor)
Urkel Tree:
An optimized and cryptographically provable key-value store for decentralized naming.
Authenticated Data Structures
WARNING:

Basic Explanation
Merkle Tree

Creation
Merkle Tree
Inclusion Proof
What about Bob Ethereum?
Ethereum Wish List

• key-value store

• allow updates w/o entire tree reconstruction

• bounded depth

• history independence
WARNING:

Another Basic Explanation
Merkle Patricia Tree

Block Header, \( H \) or \( B_c \), stateRoot, \( H_f \),
Kekack 256-bit hash of the root node of the state tree, after all
transactions are executed and finalizations applied.

Hash function:
KECCAK256()

World State Trie

ROOT: Extension Node
prefix | shared nibble(s) | next node
0      | a7              |

Branch Node

Leaf Node
prefix | key-end  | value
2      | 1355     | 45.0ETH

Extension Node
prefix | shared nibble(s) | next node
0      | d3        |

Leaf Node
prefix | key-end  | value
2      | 9365     | 1.1ETH

Prefixes
0 - Extension Node, even number of nibbles
1 - Extension Node, odd number of nibbles,
2 - Leaf Node, even number of nibbles
3 - Leaf Node, odd number of nibbles
1 nibble = 4 bits

Simplified World State, \( \sigma \)

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 7</td>
<td>1 1 3 5 5</td>
</tr>
<tr>
<td>a 7</td>
<td>7 7 d 3 3 7</td>
</tr>
<tr>
<td>a 7</td>
<td>7 f 9 3 6 5</td>
</tr>
<tr>
<td>a 7</td>
<td>7 7 d 3 9 7</td>
</tr>
</tbody>
</table>

| 45.0 ETH | a 7 1 1 3 5 5 |
| 1.00 WEI | a 7 7 d 3 3 7 |
| 1.1 ETH  | a 7 f 9 3 6 5 |
| 0.12 ETH | a 7 7 d 3 9 7 |

Ethereum Modified Merkle-Patricia-Trie System
An interpretation of the Ethereum Project Yellow Paper
© Fork Vincent C. van der Water Published September 2015
Vincent Water
What about Bob Handshake?
Handshake Wish List

- key-value store with minimal storage
- exceptional performance on SSDs
- small proof size (< 1kb)
- history independence
The Candidates

• re-balancing data structures
• Merkle Patricia Tree
• Sparse Merkle Tree
Merkle Patricia Tree

- Proof sizes too large
- Storage requirements too large
Sparse Merkle Tree

- Too many database lookups
- Too much hashing
The Decision ?
Urkel Tree:

An optimized and cryptographically provable key-value store for decentralized naming.
Urkel Tree

- Base-2 merklelized trie
- Two types of nodes: internal and leaf
- append-only files
Structure on Disk

```c
struct internal_node_s {
    uint8_t left_hash[32];
    uint16_t left_file;
    uint32_t left_position;
    uint8_t right_hash[32];
    uint16_t right_file;
    uint32_t right_position;
} internal_node;
```

```c
struct leaf_node_s {
    uint8_t key[32];
    uint16_t value_file;
    uint32_t value_position;
    uint16_t value_size;
} leaf_node;
```
Urkel Tree:

Insertion
Map:
0000 = a
Map:
0000 = a
0000 = b

R
/ \
/   \\a   b
Map:
0000 = a
1100 = b
1101 = c
Map:
0000 = a
1100 = b
1101 = c
1000 = d
Map:
0000 = a
1100 = b
1101 = c
1000 = d
1001 = e
Urkel Tree:

Removal
Map:
0000 = a
1100 = b
1101 = c
1000 = d
Map:
0000 = a
1100 = b
1101 = c
Map:
0000 = a
1100 = b
Map:

0000 = a
1100 = b
Map:
0000 = a
1100 = b
Map:
0000 = a
1100 = b

\[
\begin{array}{c}
R \\
/ \ \\
/ \\
a \\
\end{array}
\]

\[
\begin{array}{c}
b \\
\end{array}
\]
Map:

\[ \begin{array}{c}
\text{0000} = a \\
\end{array} \]
Map:

0000 = a
Urkel Tree:

Proofs
Urkel Tree:

Leaf Hash

\( \text{HASH}(0x00 || 256\text{-bit-key} || \text{HASH(value)}) \)
Map:
  0000 = a
  1100 = b
  1101 = c
  1000 = d

Proof of non-inclusion for 1110.
Map:
0000 = a
1100 = b
1101 = c
1000 = d

Proof of non-inclusion for 1110.
Proof of non-inclusion for 0100.
Map:
0000 = a
1100 = b
1101 = c
1000 = d

Proof of non-inclusion for 0100.
Map:
0000 = a
1100 = b
1101 = c
1000 = d

Proof of inclusion for 1101.
Benchmarks

- Intel Core i7-7500U with 2.7GHz
- NVMe PCIe SSD
- 50 million 300-byte leaves
- 500 leave batches
- periodic commissions of 44,000 values
Benchmarks (cont.)

- 500 value batches averaged 100-150ms
- 44,000 value commissions averaged 400-600ms
- Average node depth of 27 or 28 bits
- ~800 bytes proof size
- 1-2ms proof creation time
Merklix Variant

- Attackers can grind keys to grow tree
- Bitcoin produces 72-80 bit collisions on block headers
- base-2 merkelized radix tree for DoS protection
- adds complexity to code
Summary

Primary Advantages

• Performance
• Simplicity
• Storage
• Proof size