Ethereum

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

August 21, 2018

Ethereum

- A blockchain platform for building decentralized applications
  - Application code and state is stored on a blockchain
  - Transactions cause code execution and update state, emit events, and write logs
  - Frontend web interfaces can respond to events and read logs
- Most popular platform for creating new tokens (ICOs)
  - Each ICO implements a ERC-20 token contract (link)
  - Investments in ICOs was about $7 billion in 2017
  - About $12 billion in H1 of 2018
- Other applications
  - Ethereum Name Service (`https://ens.domains/`)
  - Cryptokitties (`https://www.cryptokitties.co/`)
  - Fomo3D (`https://fomo3d.hostedwiki.co/`)
  - Decentralized exchanges (`https://idex.market`)

Ethereum History

- Proposed by then 19 y.o. Vitalik Buterin in 2013
- VB visited the Mastercoin team in Oct 2013
- Released the Ethereum white paper in Dec 2013
- Bitcointalk announcement on Jan 24th, 2014
- A presale in July-Aug 2014 collected 31,591 BTC worth 18 million USD in return for 60,102,216 ETH
- About 12 million ETH created to pay early contributors and setup non-profit foundation
- Ethereum notable releases
  - Release 1.0: Frontier on 30 July, 2015
  - Release 2.0: Homestead on 14 March, 2016
  - Release 2.1: DAO Hard Fork on 20 July, 2016
  - Release 3.0: Metropolis phase 1, Byzantium on 16 Oct, 2017
    - Support for zkSNARKs
  - Release 3.1: Metropolis phase 2, Constantinople, expected in 2018
  - Release 4.0: Serenity, TBA
    - Move from proof-of-work to proof-of-stake

Bitcoin vs Ethereum

	Bitcoin	Ethereum
Specification	Bitcoin Core client	Ethereum yellow paper
Consensus	SHA256 PoW	Ethash PoW (later PoS)
Contract Language	Script	EVM bytecode
Block interval	10 minutes	14 to 15 seconds[1]
Block size limit	approx 4 MB	11 KB to 34 KB (Aug 2017 to Aug 2018)[2]
Difficulty adjustment	After 2016 blocks	After every block
Currency supply	Fixed to 21 million	Variable (101 million in Aug 2018)[3]
Currency units	1 BTC = 10^8 satoshi	1 ETH = 10^{18} Wei

[1] https://etherscan.io/chart/blocktime
[2] https://etherscan.io/chart/blocksize
[3] https://etherscan.io/chart/ethersupplygrowth

Ethereum Specification

- Specified in the Ethereum yellow paper by Gavin Wood
- Implemented in Go, C++, Python, Rust
- Yellow paper models Ethereum as a transaction-based state machine
  - $\boldsymbol{\sigma}_t$ = State at time t, T = Transaction, Υ = Transaction-level state-transition function

$$\boldsymbol{\sigma}_{t+1} = \Upsilon(\boldsymbol{\sigma}_t, T)$$

  - B = Block (series of transactions and other stuff), Π = Block-level state-transition function

$$\begin{aligned} \boldsymbol{\sigma}_{t+1} &= \Pi(\boldsymbol{\sigma}_t, B) \\ B &= (\cdots, (T_0, T_1, \ldots), \cdots) \end{aligned}$$

  - Ω = Block finalization state-transition function

$$\Pi(\boldsymbol{\sigma}, B) = \Omega(B; \Upsilon(\Upsilon(\boldsymbol{\sigma}, T_0), T_1) \ldots)$$

Ethereum World State

- World state consists of *accounts*
- Account types
  - **Externally owned accounts**: Controlled by private keys
  - **Contract accounts**: Controlled by contract code
- Account state
  - **nonce**: Number of transactions sent or contract-creations made
  - **balance**: Number of Wei owned by this account
  - **storageRoot**: Root hash of storage Merkle Patricia trie
  - **codeHash**: Hash of EVM code if contract account
- Mapping between account addresses and states is stored in **state database**
- Each account has a 20-byte address
  - EOA address = Right-most 20 bytes of Keccak-256 hash of public key
  - Contract address = Right-most 20 bytes of Keccak-256 hash of `RLP`([senderAddress, nonce])

Keccak-256

- Cryptographic hash function used by Ethereum
- NIST announced competition for new hash standard in 2006
- Keccak declared winner in 2012
- In August 2015, FIPS 202 “*SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*” was approved
- Ethereum adopted Keccak-256 but NIST changed the padding scheme
- Keccak-256 and SHA3-256 give different outputs for the same message
    - `https://ethereum.stackexchange.com/questions/550/which-cryptographic-hash-function-does-ethereum-use`

Transactions

- Two types
  - Contract creation
  - Message calls
- Contract creation transactions create new contracts on the blockchain
  - Destination address is null
  - EVM code for account initialization is specified
- Message call transactions call methods in an existing contract
  - Input data to contract methods is specified
- Transaction execution modifies the state database

Storage Contract

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

https://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html#storage

Recursive Length Prefix Encoding

Recursive Length Prefix Encoding (1/3)

- Applications may need to store complex data structures
- RLP encoding is a method for serialization of such data
- Value to be serialized is either a byte array or a list of values
  - Examples: “abc”, [“abc”, [“def”, “ghi”], [“”]]

$$\texttt{RLP}(\mathbf{x}) = \begin{cases} R_b(\mathbf{x}) & \text{if } \mathbf{x} \text{ is a byte array} \\ R_l(\mathbf{x}) & \text{otherwise} \end{cases}$$

- $\texttt{BE}$ stands for big-endian representation of a positive integer

$$\texttt{BE}(x) = (b_0, b_1, ...) : b_0 \neq 0 \land x = \sum_{n=0}^{n<\|\mathbf{b}\|} b_n \cdot 256^{\|\mathbf{b}\|-1-n}$$

Recursive Length Prefix Encoding (2/3)

- Byte array encoding

$$R_b(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if} \quad \|\mathbf{x}\| = 1 \wedge \mathbf{x}[0] < 128 \\ (128 + \|\mathbf{x}\|) \cdot \mathbf{x} & \text{else if} \quad \|\mathbf{x}\| < 56 \\ \left(183 + \left\|\mathtt{BE}(\|\mathbf{x}\|)\right\|\right) \cdot \mathtt{BE}(\|\mathbf{x}\|) \cdot \mathbf{x} & \text{else if} \ \left\|\mathtt{BE}(\|\mathbf{x}\|)\right\| \leq 8 \end{cases}$$

- $(a) \cdot (b) \cdot c = (a, b, c)$
- Examples
  - Encoding of 0xaabbcc = 0x83aabbcc
  - Encoding of empty byte array = 0x80
  - Encoding of 0x80 = 0x8180
  - Encoding of “Lorem ipsum dolor sit amet, consectetur adipisicing elit” = 0xb8, 0x38, ’L’, ’o’, ’r’, ’e’, ’m’, ’ ’, . . . , ’e’, ’l’, ’i’, ’t’
- Length of byte array is assumed to be less than 256^8
- First byte can be at most 191

Recursive Length Prefix Encoding (3/3)

- List encoding of $\mathbf{x} = [\mathbf{x}_0, \mathbf{x}_1, \ldots]$

$$R_l(\mathbf{x}) = \begin{cases} (192 + \|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{if} \quad \|s(\mathbf{x})\| < 56 \\ (247 + \|\mathtt{BE}(\|s(\mathbf{x})\|)\|) \cdot \mathtt{BE}(\|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{otherwise} \end{cases}$$

$$s(\mathbf{x}) = \mathtt{RLP}(\mathbf{x}_0) \cdot \mathtt{RLP}(\mathbf{x}_1) \ldots$$

- Examples
    - Encoding of empty list [ ] = 0xc0
    - Encoding of list containing empty list [[ ]] = 0xc1 0xc0
    - Encoding of [[ ], [[ ]], [[ ], [[ ]]]] = 0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0
- First byte of RLP encoded data specifies its type
    - 0x00, . . . , 0x7f $\Longrightarrow$ byte
    - 0x80, . . . , 0xbf $\Longrightarrow$ byte array
    - 0xc0, . . . , 0xff $\Longrightarrow$ list

Reference: `https://github.com/ethereum/wiki/wiki/RLP`

Merkle Patricia Trie

Merkle Trie

- A trie is a search tree with string keys
- Example: Trie with hexadecimal string keys
  - Every node is of the form $[i_0, i_1, \ldots, i_{15}, \text{value}]$
  - Consider key-value pairs: ('do', 'verb'), ('dog', 'puppy'), ('doge', 'coin'), ('horse', 'stallion')
  - What is the corresponding radix tree?
- Merkle tries are a cryptographically secure data structure used to store key-value bindings
  - Instead of pointers, the hash of a node is used for lookup in a **database**
  - Location of `node` in database is at key `Hash(RLP(node))`
- $\mathcal{O}(\log N)$ Merkle proofs showing the existence of a leaf in a trie with given root hash

Merkle Trie Update

```python
# Update value at path in a trie with root hash equal to node_hash
def update(node_hash, path, value):
    # Get the node with key node_hash from database
    # If it does not exist, create a new NULL node
    curnode = db.get(node_hash) if node else [NULL]*17
    newnode = curnode.copy()

    if path == '':
        # If end of path is reached, insert value in current node
        newnode[-1] = value
    else:
        # Update node indexed by first path nibble and proceed
        newindex = update(curnode[path[0]], path[1:], value)
        # Update hash value of node indexed by first path nibble
        newnode[path[0]] = newindex

    # Insert database entry with hash-node key-value pair
    db.put(hash(newnode), newnode)
    return hash(newnode)
```

Source: `https://github.com/ethereum/wiki/wiki/Patricia-Tree`

Merkle Patricia Trie

- Merkle tries are inefficient due to large number of empty nodes
- PATRICIA = Practical Algorithm To Retrieve Information Coded in Alphanumeric
- Node which is an only child is merged with its parent
- A node in a Merkle Patricia trie is either
  - **NULL**
  - **Branch**: A 17-item node $[i_0, i_1, \ldots, i_{15}, \texttt{value}]$
  - **Leaf**: A 2-item node $[\texttt{encodedPath}, \texttt{value}]$
  - **Extension**: A 2-item node $[\texttt{encodedPath}, \texttt{key}]$
- In leaf nodes, `encodedPath` completes the remainder of a path to the target `value`
- In extension nodes
  - `encodedPath` species partial path to skip
  - `key` specifies location of next node in database
- Two requirements
  - Need some way to distinguish between leaf and extension nodes
  - `encodedPath` is a nibble array which needs to be byte array

Hex-Prefix Encoding

- Efficient method to encode nibbles into a byte array
- Also stores an additional flag t
- Let $\mathbf{x} = [\mathbf{x}[0], \mathbf{x}[1], \ldots,]$ be a sequence of nibbles

$$
\begin{aligned}
\texttt{HP}(\mathbf{x}, t) &= \begin{cases} (16f(t), 16\mathbf{x}[0] + \mathbf{x}[1], 16\mathbf{x}[2] + \mathbf{x}[3], ...) & \text{if } \|\mathbf{x}\| \text{ is even} \\ (16(f(t) + 1) + \mathbf{x}[0], 16\mathbf{x}[1] + \mathbf{x}[2], 16\mathbf{x}[3] + \mathbf{x}[4], ...) & \text{o.w.} \end{cases} \\
f(t) &= \begin{cases} 2 & \text{if} \quad t \neq 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

- High nibble of first byte has two bits of information
  - Lowest bit encodes oddness of length
  - Second-lowest bit encodes the flag
- Low nibble of first byte is zero if length is even and equal to first nibble otherwise

Hex-Prefix Encoding of Trie Paths

- First nibble of `encodedPath`

Hex	Bits	Node Type	Path Length
0	0000	extension	even
1	0001	extension	odd
2	0010	leaf	even
3	0011	leaf	odd

- Examples
  - $[0, f, 1, c, b, 8, \text{value}] \rightarrow$ '20 0f 1c b8'
  - $[f, 1, c, b, 8, \text{value}] \rightarrow$ '3f 1c b8'
  - $[1, 2, 3, 4, 5, \ldots] \rightarrow$ '11 23 45'
  - $[0, 1, 2, 3, 4, 5, \ldots] \rightarrow$ '00 01 23 45'

Example Merkle Patricia Trie

- Key-value pairs: (‘do’, ‘verb’), (‘dog’, ‘puppy’), (‘doge’, ‘coin’), (‘horse’, ‘stallion’)
- Hex keys and their values
  - 64 6f : ‘verb’
  - 64 6f 67 : ‘puppy’
  - 64 6f 67 65 : ‘coin’
  - 68 6f 72 73 65 : ‘stallion’
- Trie

rootHash	[<16>, hashA]
hashA	[<>, <>, <>, <>, hashB, <>, <>, <>, hashC, <>, <>, <>, <>, <>, <>, <>, <>]
hashC	[<20 6f 72 73 65>, ’stallion’]
hashB	[<00 6f>, hashD]
hashD	[<>, <>, <>, <>, <>, <>, hashE, <>, <>, <>, <>, <>, <>, <>, <>, <>, ’verb’]
hashE	[<17>, hashF]
hashF	[<>, <>, <>, <>, <>, <>, hashG, <>, <>, <>, <>, <>, <>, <>, <>, <>, ’puppy’]
hashG	[<35>, ’coin’]

References

- White paper https://github.com/ethereum/wiki/wiki/White-Paper
- Ethereum Wikipedia Article https://en.wikipedia.org/wiki/Ethereum
- A Prehistory of the Ethereum Protocol https://vitalik.ca/general/2017/09/14/prehistory.html
- Ethereum announcement on Bitcointalk https://bitcointalk.org/index.php?topic=428589.0
- History of Ethereum http://ethdocs.org/en/latest/introduction/history-of-ethereum.html
- The DAO Wikipedia Article https://en.wikipedia.org/wiki/The_DAO_(organization)
- Releases https://github.com/ethereum/wiki/wiki/Releases
- Yellow paper https://ethereum.github.io/yellowpaper/paper.pdf
- Merkle Patricia Tree https://github.com/ethereum/wiki/wiki/Patricia-Tree