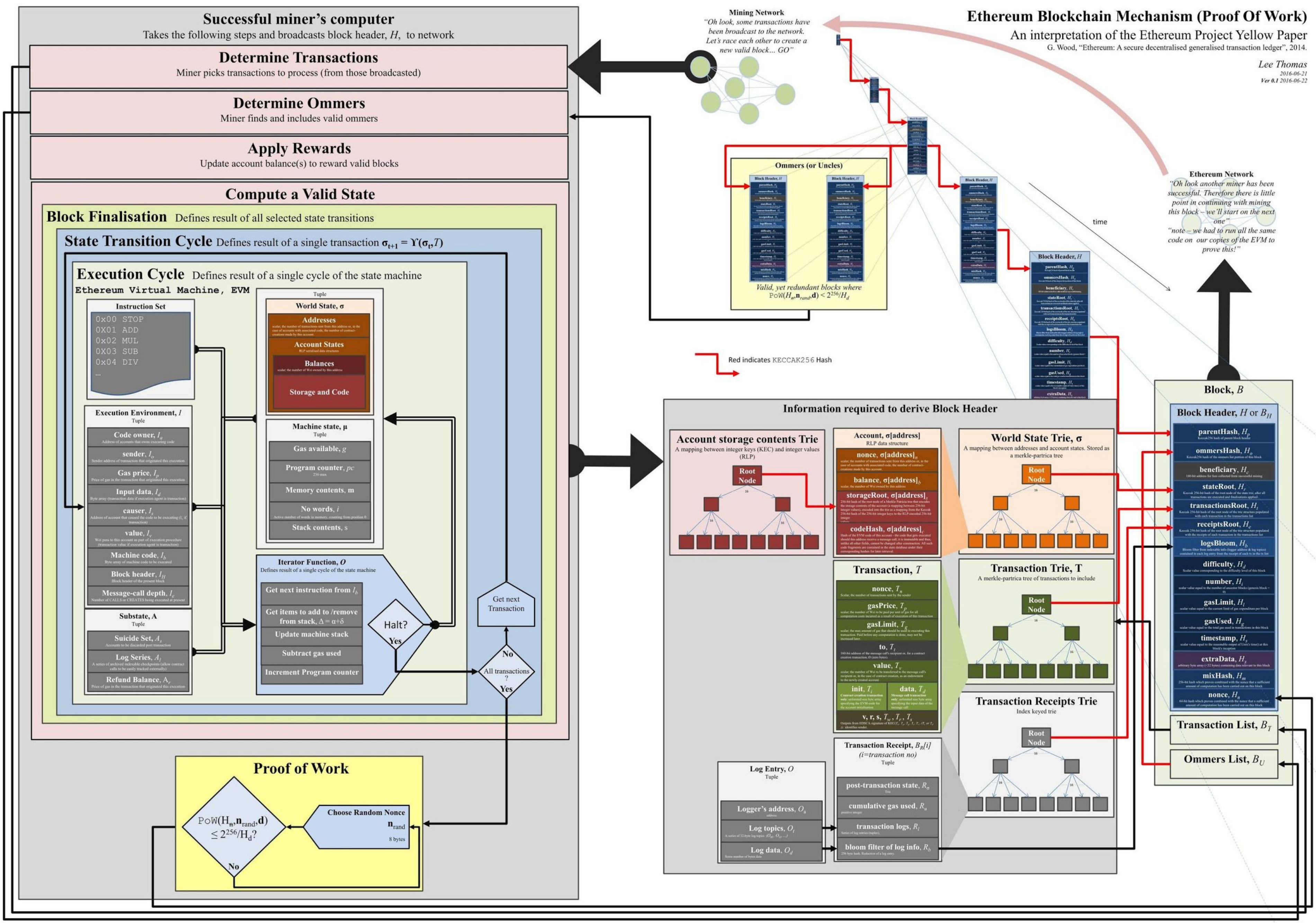
Technical dive into Ethereum


Ethereum Blockchain Mechanism (Proof Of Work)
An interpretation of the Ethereum Project Yellow Paper
G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.
Lee Thomas
2016-06-21
Ver 0.1 2016-06-22
Mining Network
"Oh look, some transactions have been broadcast to the network. Let's race each other to create a new valid block... GO"
Ethereum Network
"Oh look another miner has been successful. Therefore there is little point in continuing with mining this block – we'll start on the next one"
"note – we had to run all the same code on our copies of the EVM to prove this!"
time
Red indicates KECCAK256 Hash
Ommers (or Uncles)
Valid, yet redundant blocks where
Successful miner's computer
Takes the following steps and broadcasts block header, H, to network
Determine Transactions
Miner picks transactions to process (from those broadcasted)
Determine Ommers
Miner finds and includes valid ommers
Apply Rewards
Update account balance(s) to reward valid blocks
Compute a Valid State
Block Finalisation Defines result of all selected state transitions
State Transition Cycle Defines result of a single transaction σt+1 = Υ(σt,T)
Execution Cycle Defines result of a single cycle of the state machine
Ethereum Virtual Machine, EVM
Instruction Set
0x00 STOP
0X01 ADD
0x02 MUL
0X03 SUB
0x04 DIV
...
Execution Environment, I
Tuple
Code owner, Ia
sender, Io
Gas price, Ip
Input data, Id
causer, Is
value, Iv
Machine code, Ib
Block header, IH
Message-call depth, Ie
Substate, A
Tuple
Suicide Set, As
Log Series, Al
Refund Balance, Ar
Tuple
World State, σ
Addresses
Account States
Balances
Storage and Code
Machine state, μ
Tuple
Gas available, g
Program counter, pc
Memory contents, m
No words, i
Stack contents, s
Iterator Function, O
Defines result of a single cycle of the state machine
Get next instruction from Ib
Get items to add to /remove from stack, Δ = α+δ
Update machine stack
Subtract gas used
Increment Program counter
Halt?
Yes
Get next Transaction
No
All transactions ?
Yes
Proof of Work
Choose Random Nonce
nrand
8 bytes
PoW(Hn,nrand,d) ≤ 2256/Hd?
No
Information required to derive Block Header
Account storage contents Trie
A mapping between integer keys (KEC) and integer values (RLP)
Root Node
Account, σ[address]
RLP data structure
nonce, σ[address]n
balance, σ[address]b
storageRoot, σ[address]s
codeHash, σ[address]c
World State Trie, σ
A mapping between addresses and account states. Stored as a merkle-partrica tree
Root Node
Transaction, T
nonce, Tn
gasPrice, Tp
gasLimit, Tg
to, Tt
value, Tv
init, Ti
data, Td
v, r, s, Tw, Tr, Ts
Transaction Trie, T
A merkle-partrica tree of transactions to include
Root Node
Transaction Receipts Trie
Index keyed trie
Root Node
Log Entry, O
Tuple
Logger's address, Oa
Log topics, Ot
Log data, Od
Transaction Receipt, BR[i]
(i=transaction no)
Tuple
post-transaction state, Rσ
cumulative gas used, Ru
transaction logs, Rl
bloom filter of log info, Rb
Block, B
Block Header, H or BH
parentHash, Hp
ommersHash, Ho
beneficiary, Hc
stateRoot, Hr
transactionsRoot, Ht
receiptsRoot, He
logsBloom, Hb
difficulty, Hd
number, Hi
gasLimit, Hl
gasUsed, Hg
timestamp, Hs
extraData, Hx
mixHash, Hm
nonce, Hn
Transaction List, BT
Ommers List, BU

What are Ethereum Account?

Two types of accounts:

Externally Owned Accounts

0x995a30e66ff7c050c7fdb065e13a83bcf02ba559

- Nonce
- Current Balance
- Code: null
- Storage: null
- Only controlled by private key

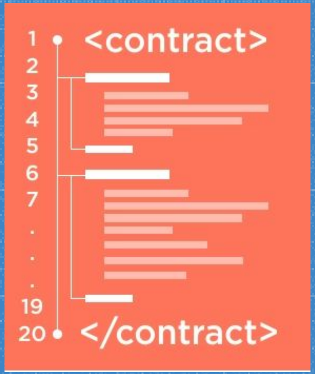
Pros:

- Simple
- Light in size
- Predictable

Cons:

- Not customizable
- Cannot see incoming txns

Contract Accounts



- Nonce
- Current Balance
- Contract Code
- Storage
- Controlled only by its code
- Every time it receives a txn, the code is run

Pros:

- Highly Customizable
- Can see all transactions to/from

Cons:

- Can be unsafe
- Bigger blockchain

Transactions vs Messages

Two different ways to interact between accounts:

Transactions:

From EOA to anyone

- **Signature** of the sender
- The recipient of the message (**to**)
- The amount of ether to transfer from the sender to the recipient (**balance**)
- An optional **data** field
- A STARTGAS value
- A GASPRICE value

Messages:

From Contract to other Contracts, think function calls

Contains:

- Same as txn **except** for the signature

They are:

- Never serialized (no transfer over the wire)
- Exists only in the EVM execution environment
- Lead other contract’s code to run
- Done using the CALL or DELEGATECALL Opcode

Note: Messages also need gas.

To initiate anything in Ethereum, everything starts from an EOA transaction (which is signed, of course)

Inside an Ethereum Block Header

```
{
  difficulty: 427164, // difficulty of the POW for the Block
  extraData: "0xd783010600846765746887676f312e382e31856c696e7578", // here the type of node (geth v1.8)
  gasLimit: 4712388,
  gasUsed: 748070,
  hash: "0xab7f5a4c47e4097af83a3306235a6867243ad69bc15196063b4beedf7968c612", // hash of the block
  logsBloom:
"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000", // OR bloom filter of all the txn receipt blooms
  miner: "0x1cc1f899d8ad034febd1c0bd65fa121fce1b28d3", // miner of the block
  mixHash: "0xc9fb17996bef2d159f6ad4877e79ea303e0d80ee1846a5b362385da6e7c79678", // use with nonce for Ethash
  nonce: "0x7b15aa634a173202", // nonce for the PoW
  number: 98230, // block#
  parentHash: "0x930fd9f70db8b35481a9ec51ef1254d40aa6a1745aa38eb329d4b02dfff1850f", // chain!!
  receiptsRoot: "0x0e44542f06c5ee5f6efaa048380895811833807195e0c161e5b82a250b118d6e", // root of the receipts trie
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347", // sha3 of uncles data in block
  size: 3171, // size of the block in bits
  stateRoot: "0x98c79928e3458f86d6d455aca70571b9a9f40e389d3271dac8df8e08c3ba7fb5", // root of the state trie
  timestamp: 1496187844, // unix timestamp
  totalDifficulty: 43078242275, // total of all the difficulties in the chain
  transactions: ["0x20395aa8a8148fb2f28dc105e6f100b842615ea7ab666bc7a05ad1d8c146d0b2"], // list of txns**
  transactionsRoot: "0xde86b85e333c62ba9c8bd09c035e96a7f4b2081df54a2a8003a4dfede5cbab65", // root of txn trie
  uncles: [] // list of uncles
}
```

** under **transactions** we only see the transaction hash (for brevity) but really it is a list of transaction objects.

Inside an Ethereum Transaction

```
{
  blockHash: "0x193007f42b336218d24acd5b285c37276dfca4ce370e93af5c6684fa1e8ac25f", // hash of the block
  blockNumber: 50306, // block#
  from: "0x32cd3282d33ff58b4ae8402a226a0b27441b7f1a", // from address
  gas: 940000, // gas inputted
  gasPrice: 20000000000, // willing price to pay per computation (remember gas*gasPrice = amount of Wei to pay)
  hash: "0x23cb3a93f8ce210f3bf94e9014e94fe36f5968910335c4bbdcce03061fdcdc59", // hash of the txn
  input:
"0xd441987d00000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000000000000000
00000000000000000000c0000000000000000000000000000000000000000000000000000000000000000300000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000
000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000003000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
090000000000000000000000000000000000000000000000000000000000000008", // This looks like a function invocation w/
parameters
  nonce: 10, // number of txns this account (from) has done
  r: "0xe988264608a6df786ab0aa2457005cadc0b83dee991317a451f5704e2da8b8fc", // ECDSA r value
  s: "0x7e07a0de40da7a7fd2314acd85fb995e4f73171570c09ebe46f6115c5117d92a", // ECDSA s value
  to: "0x6871a7fa6ab6048388c35486399259fecb836f37", // to address (null if creating a contract)
  transactionIndex: 0, // position in the txn list in the Block (matters!)
  v: "0x2a", // ecrecover value
  value: 0 // amount sent along with the txn
}
```

Inside an Ethereum Receipt

```
{
  blockHash: "0xc7e48b08c038040b561c57a1a4ff7bfcf9cac7b1a37399f7d1fa5d9cabe7ee01", // hash of the block
  blockNumber: 70667, // block#
  contractAddress: null, // address of the contract if this txn created a contract
  cumulativeGasUsed: 98571, // cumulative gas used in the block
  from: "0x13c0127b66b336a644cc36c2e44eadc5fcd8b79d", // sum of gasUsed by this txn & preceding ones in this block
  gasUsed: 98571, // gas used for this transaction only
  logs: [{
      address: "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38", // address from which the log originated
      blockHash: "0xc7e48b08c038040b561c57a1a4ff7bfcf9cac7b1a37399f7d1fa5d9cabe7ee01", // hash of the block
      blockNumber: 70667, // block#
      data:
"0x45544855534400000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000", // contains one or more 32 Bytes non-indexed arguments of the log
      logIndex: 0, // position of the log
      removed: false, //  true if log removed (due to chain reorg) false if it is a valid log
      topics: ["0x5a690ecd0cb15c1c1fd6b6f8a32df0d4f56cb41a54fea7e94020f013595de796", // sha3 hash of the event (in
hex) "0x0000000000000000000000000000000000000000000000000000000000000002",
"0x000000000000000000000000fc6225fc17c01c05b7d50284faab7ee1ba0546ab",
"0x0000000000000000000000000000000000000000000000000000000000000000"], // indexed logged arguments
      transactionHash: "0x79835f5376cd33a51a1f7643af8c97e74272e4935d67e7edd465737dd83db38e", // hash of the txn
      transactionIndex: 0 // txn index position
  }],
  logsBloom:
"0x04002000000000000000000000000200005000000000000000000000000000000000000000000000000000000000000200000004000200000000
001000000000000000000000000000000000000000000000000000000000000000000000000000000000000002040000000008000000800400000
000000010000000000000000000000088000000200000000000000000000000000000000000000000000000081000000000000000000001000000
000000000000000000000000000004000000000000000000000200000000000000000000000000000000000000000000020000000000000000000
0000000000000000800000000800000000000000000000", //bloom filter, here in hex, not binary
  root: "0x32e...", //hash of the root of the intermediate state trie after the txn has been applied
  to: "0xfc6225fc17c01c05b7d50284faab7ee1ba0546ab", // to address
  transactionHash: "0x79835f5376cd33a51a1f7643af8c97e74272e4935d67e7edd465737dd83db38e", // txn hash
  transactionIndex: 0 // txn index
}
```

A little word on **nonces**

Used in many aspects in Ethereum:

- PoW Nonce (random number used with mixHash to prove computation)
- Contract Nonce
- EOA Nonce

Rules for incrementations

- If an externally-owned account sends a transaction, its nonce is incremented **before** execution
- If an externally-owned account creates a contract, its nonce is incremented **before** execution
- If a contract sends a message, **no nonce increments happen**
- If a contract creates a contract, the nonce is incremented **before** the rest of the sub-execution
- The pre-increment nonce is used to determine the contract address
- Nonce increments are never **reverted**

`

Technical Design

Keccak256 Hashing Algorithm:

The Keccak hashing algorithm is part of the SHA3 family (whereas SHA256, used in bitcoin, is part of the SHA-2 family). It was created by Guido Bertoni, Joan Daemen, Michael Peeters and GIlles Van Asshche as part of the NIST hash function competition, organised with the goal to find a new hashing algorithm as it was believed at the time that SHA-2 was potentially reversible.

Ethereum started using it right away and then a new standard, the FIPS-202, changed a some constants in the implementation (August 2015) which was called. FIPS-202 was officially called SHA3.

So SHA3 in ethereum **is not** the same SHA3 in the standards!

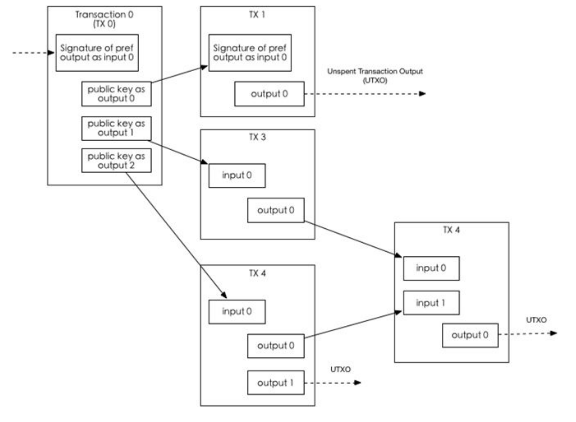
Ethereum address generation && ECDS choice:

1) Same Elliptic curve as bitcoin secp256k1 $y^2 = x^3 + 7$
2) Create private key (d)
3) Derive public key Q = d*P, Q is the public key, P is the base Point (known)
4) The address is the **160 rightmost bits** of the Keccak256 hash of the public key
   Address = Keccak(publicKey)[96..255]
5) Add 0x to the beginning (for Hex purposes) - 42 chars - 21 bytes
6) Subsequent addresses (contract created from this account) are the hash of the address with the (pre incremented) nonce.

Account Balances, no UTXOs

Bitcoin’s “State” are all the UTXO’s

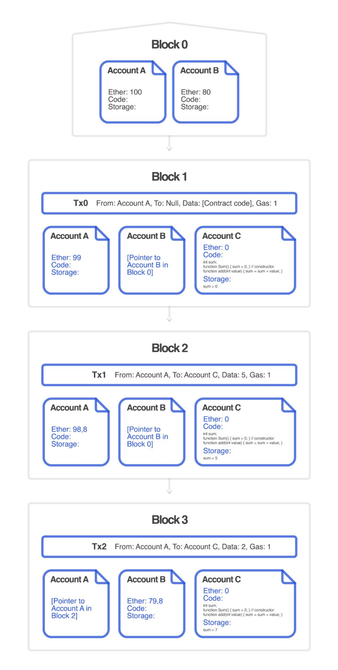
- Each coin has an owner and a value
- Every referenced input must be valid and not yet spent
- The transaction must have a signature matching the owner of the input for every input
- The total value of the inputs must equal or exceed the total value of the outputs
- The balance is all the UTXO’s attached to an owner



Transaction brief

Ethereum: Simple Balances

- Space Savings (1 UTXO = 20 + 32 + 8 = 60 bytes!!)
- Greater fungibility
- Simple
- Best for light clients
- Contracts don’t need Pub/Priv key pair
- Specific coins are not traceable, only account relationships



The Bitcoin Block Time Problem

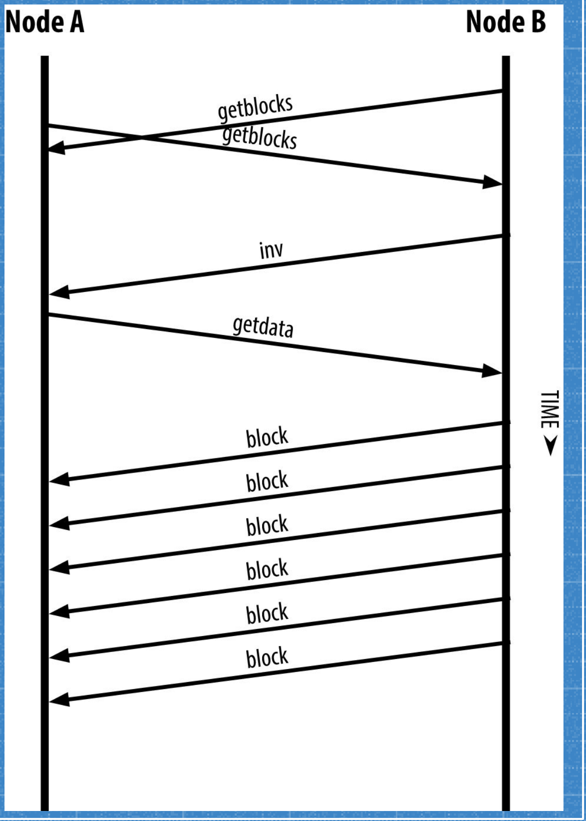
Multiple problems are trying to be solved:

1) 10 minutes block time is too slow
2) BUT we have a way shorter block time, stale rate goes up
3) AND stale rate goes up, network **efficiency** and **security** go down
4) ALSO there is the problem of centralisation of power through mining pools (>50% computing power in 2014 GHash.io in Bitcoin)

First, how do blocks propagate?

- Blocks are not forwarded directly
- Node sends an **INV** message to other nodes once block has been completely verified
- **INV** message contains:
  - Set of transaction hashes available
  - Set of and block hashes available

Upon noticing a missing element, the receiving node will make a **getData** message containing the missing hashes



Bitcoin Block Propagation time

Block propagation times (Bitcoin):

Stats:

- 6.5 seconds for blocks to propagate to 50% of nodes
- 40 seconds for 95%
- Mean delay of **12.6** seconds

Assuming

- Propagation_time = 12.6 seconds (others in the network are essentially "stagnant" for 12.6s)
- fork_rate = 1.69% (actual)
- Block Time of ~600s

Effective computational power of the network: 1 - 12.6/600 = 0.979 or 97.9% computing power

Thus, to launch a 51% attack, you only need 48.95% computing power. (w/ block time = 60s → 40% computing power!!)

Note: the block propagation time is dependant on the block size, blocks in ethereum are way smaller

As an aside:

Chances of a txn being confirmed in the real world for Bitcoin

- *63.4% t < 10 minutes*
- *13.5% 10 < t < 20 mins*
- *0.35% t > 60 mins*

Other Problem: Mining Pools

Centralization of the network

- Power to censor
- Decision makers in the space

Pool always has an edge

People in the pool will know blocks have been mined faster (since they send messages directly to the participants)

- They thus have less unused computation
- Higher efficiency compared to everyone else (depends on block propagation time && size)

51% Attacks very possible with mining pools

- GHASH.io in 2014

Solution: GHOST Protocol (By Aviv Zohar and Yonatan Sompolinsky)

Big Idea:

- Count the stale blocks (uncles/ommers) as part of the total weight (totalDifficulty in the blockHeader) of the chain

Effects:

- Main chain can hypothetically go down to 5% efficiency
- The attacker will need to outweigh the entire network for a 51% attack (not only mine new blocks but also uncle blocks to get above that difficulty)
- Solves efficiency all the way to 1 second blocks

But it doesn't solve the centralization problem, as a sizeable mining pool could mine 100% of the blocks, (remember, they have the block propagation advantage) thus getting **100% of the rewards.**

Ethereum Solution: **incentivize the uncles**!

If you incentivize the uncles, such that the miners get at maximum their percentage of hashing power as reward, then it doesn't matter so much who is the first one to create a block, what matters is that it is valid. So a 30% miner will get (a bit more than) 30% of the reward.

Logic:

- Every block points to a parent, and can have zero or more uncles.
- Uncle: block w/ valid header which is the child of the parent's parent.
- Actual Block reward: 1*blockReward + n*(1/32), n being # of uncles included
- Miner of the uncle block gets 7/8 * blockReward
- Maximum of 7th generational uncle
- No 2 of the same uncles in the 7 blocks
- Only 2 uncles per block
- "Score" of the block is 0 @ genesis, and then totalDifficulty = parentScore + difficulty*(1 + uncleCount)

Gas and fees

Halting Problem! (woot CS!!)

The EVM being Turing complete, we need a way to halt its computation

- When creating a transaction, you provide 2 things:
  - The max amount of computation that you want your program to run (gas)
  - The price per computation you are willing to pay (gasPrice) in Wei (10^-18 eth)

The total you will be paying for a computation will be gas*gasPrice.

Now how does one know how much to pay?

- Every OPCODE has an associated price (which turns out, if not thought of carefully, can lead to DDoS'ing the network)
- 21000 gas for any transaction as a base fee. Covers the cost of the elliptic curve operation to recover the sender's address from the signature as well as the disk bandwidth to store the transaction.
- You can 'fake run' your txn locally to get an estimate at the amount of gas used (not accurate though)

Warning:

- As transactions are atomic, if they run out of gas (Out of Gas exception), everything that happened in the transaction will be reverted (state change etc). Your txn will fail and the miner keeps **ALL** of the gas sent
- If you send too much, you get refunded what wasn't needed

Also, there is a total **block gas limit** that one cannot exceed (currently ~470000)

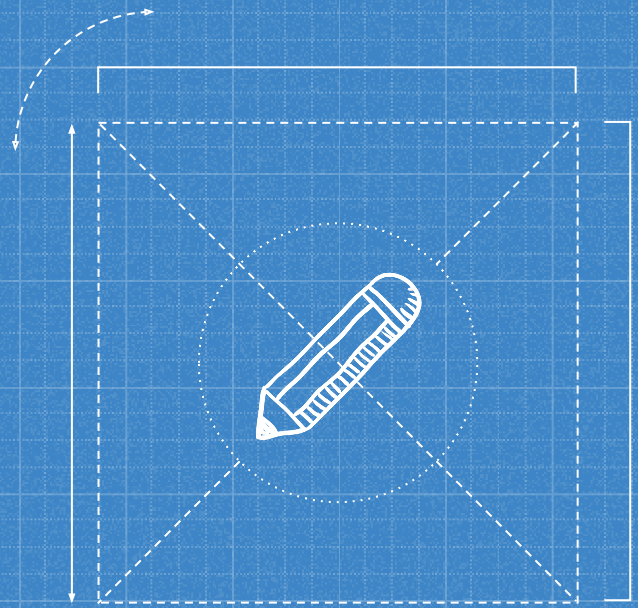
- Keeps the blocks small (good for propagation time)
- Keeps execution time fast (remember, miners need to first apply all txns)

Custom Features

- Merkle Patricia Tries
  - Bound to depth of 40
  - O(log(n)) for lookup
  - O(log(n)) for update, insert, delete!!
  - Merkle Proofs for light clients

- RLP serialization
  - Minimalist serialization
  - Easy to implement
  - Safe/guaranteed absolute byte-perfect consistency

- Bloom Filters
  - Efficiently query if txns are members of a block
  - Same for receipt
- EVM
  - Most awesome small computer



The Ethereum Virtual Machine

“The billion dollar computer” -me

EVM

- Stack based virtual machine
- Elements on the stack are 32 byte words
- Same for key/value storage
- Max stack size: 1024
- Call depth limit 1024 (don’t do recursion!!)
- Every computational step taken in a program's execution must be paid for up front to prevent DDOS.
- No access to other contract’s storage/state
  - Interaction only done through messages where data can be passed (arbitrary length byte array)
- Program execution is sandboxed
- Program execution is fully deterministic and produces identical state transitions for any conforming implementation beginning in an identical state.
- The code for the EVM is in the contract accounts. Upon sending a transaction to a contract, the EVM will do a CALLCODE operation to retrive the code in the corresponding address

When creating a contract, you actually send a transaction to the “0” address with the relevant data in the data field. This data is not the code of the contract itself, it is the code that will be run by the EVM and **return** the contract code (it will use the SSTORE OPcode to store the bytecode in the “code” field of the account tuple).

Merkle Patricia Tries

Merkle Patricia Tries

Merkle tree

- Tree where all the leaf nodes are hashed together all the way to a root node
- Allows for merkle proofs (authenticate a large dataset using a small amount of data)

Patricia Tries

- PATRICIA: Practical Algorithm To Retrieve Information Coded In Alphanumeric
- Efficient way to store data (great for dictionaries!)
- The key to access the value is actually the path taken from the root node
- Bounded to the maximum length of the key

For txn and receipts trees, they don't need to be MPT's (even though they are):

- Take all the txns in the txn list (they are the **leafs** in the trie)
- Hash all of the txns up to the root node
- Get the root is published on the blockchain

Ethereum's State Implementation, merge both ideas

- Every node parent node is the hash of the children nodes
- The path down to the value is the address (as it is hex, 16 possible values)
- The "value" is the tuple (balance, code, nonce, storage)

Optimizations:

- Different types of nodes:
    - NULL node
    - Leaf node (k,v node)
    - Extension node (k,v node but value is hash of another node)
    - Branch node (list of length 17, first 16 chars are possible hex, final element can hold the k,v node that stops there)
    - Store the node as close to the root as possible

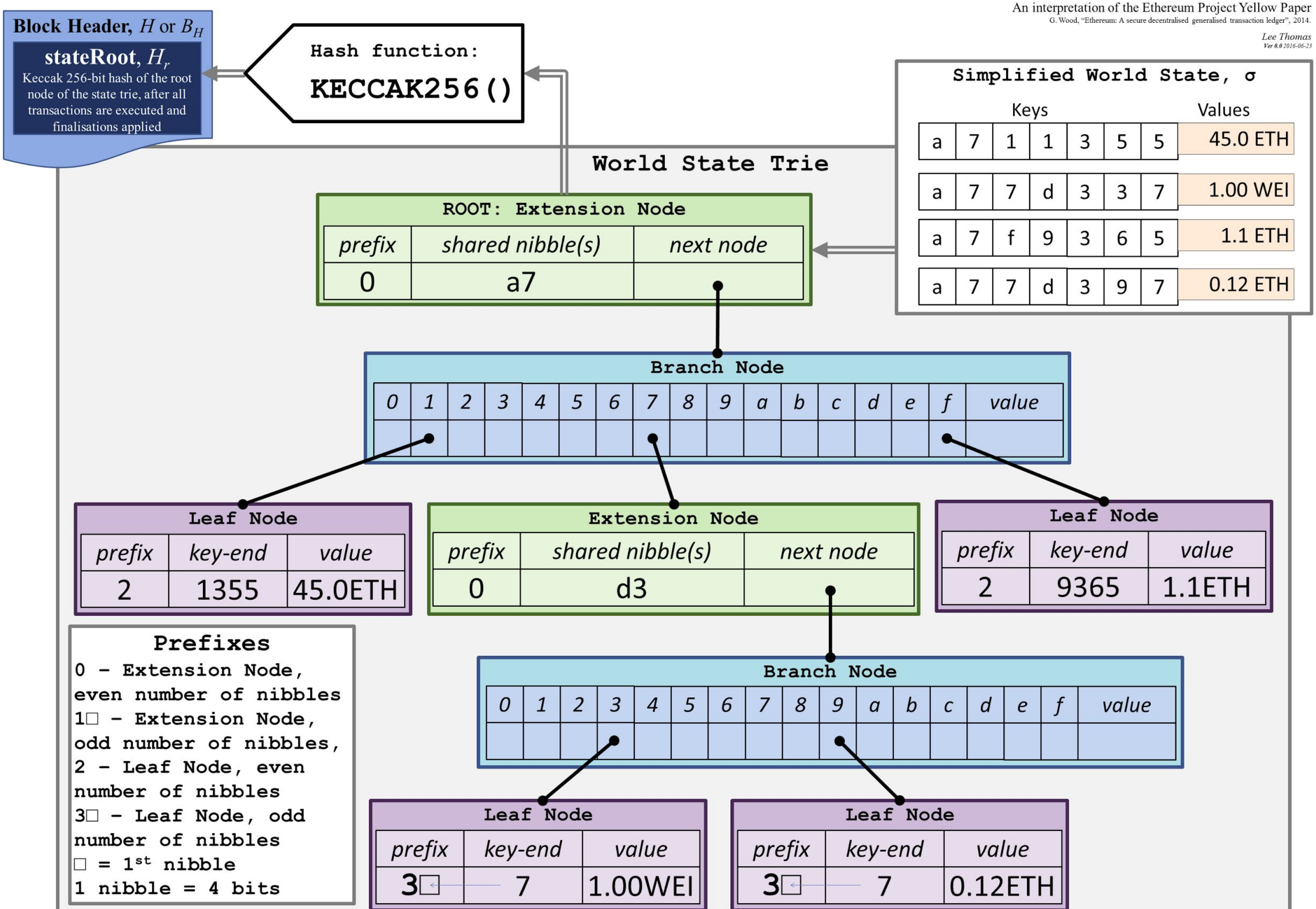
Ethereum Modified Merkle-Paricia-Trie System

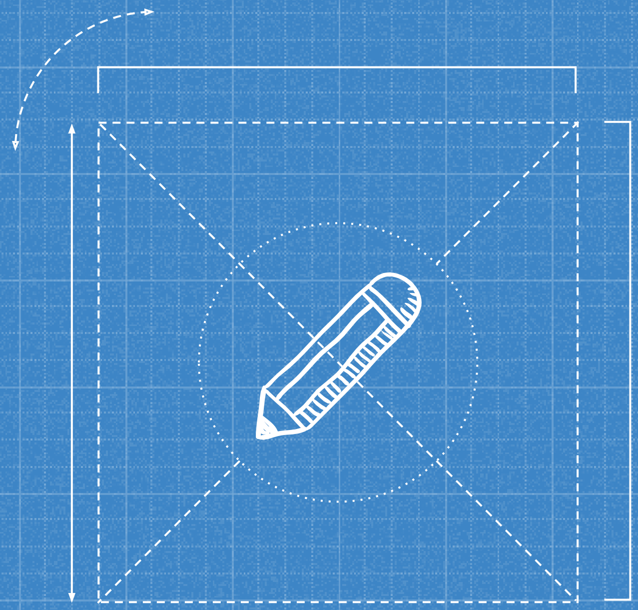
An interpretation of the Ethereum Project Yellow Paper

G. Wood, “Ethereum: A secure decentralised generalised transaction ledger”, 2014.

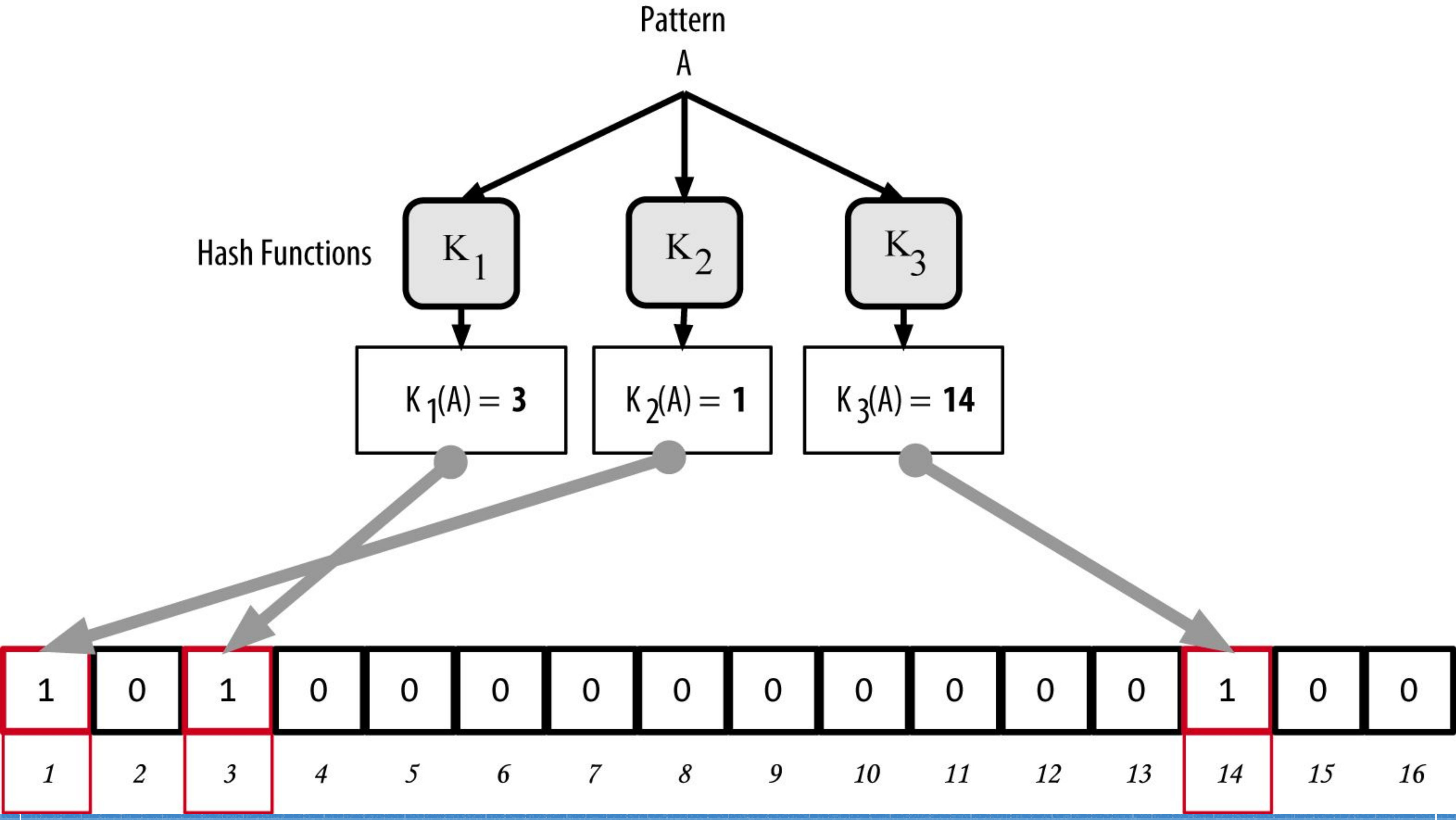
Lee Thomas

Ver 0.0 2016-06-23





Bloom Filters


Pattern
A
Hash Functions
K_1
K_2
K_3
$K_1(A) = \mathbf{3}$
$K_2(A) = \mathbf{1}$
$K_3(A) = \mathbf{14}$
1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Difficulty Bomb

As this difficulty bomb is activated on the network, the mining difficulty will skyrocket and eventually make Ethereum mining unfeasible and extremely unprofitable.

- To switch to PoS
- Supposed to be from block 200 000 (in 2015), now delayed
- Block times could be as high as 14 minutes by 2025.

```
block_diff = parent_diff + parent_diff // 2048 *
      max(1 - (block_timestamp - parent_timestamp) // 10, -99) +
      int(2**((block.number // 100000) - 2))
```