Distributed Systems
236351
Tutorial 8
Bitcoin

Yehontan Buchnik
(Based on Dolev Adas slides)

Technion

December 20, 2018
Overview

- An electronic distributed cash system
- An unstructured peer-to-peer network
- Forms a **Blockchain**
  - Public history of transactions
- Untrusted model
  - Holds as long at least 51% of the network’s CPU power are honest (rather than 51% of the participants)
  - Assumes synchronous network
  - Singed messages
  - Proof of work (PoW)
- Unknown author (Satoshi Nakamoto)
- [https://bitcoin.org/bitcoin.pdf](https://bitcoin.org/bitcoin.pdf)
Blockchain

Overview

- A chain of blocks
  - Each block contains a batch of transactions
- Denote by \( b_i \) the \( i_{th} \) block and by \( T_i \) the set of \( b_i \)'s transactions
  - The publisher of \( b_i \) also publish \( h(b_i) = hash(T_i, h(b_{i-1})) \) forming a chain of hashes
  - \( h(b_i) \) encapsulate the history of the chain with respect to \( b_i \)
  - Typically, \( h(b_i) \) is stored in \( b_i \)'s header alongside other metadata
- Forging an existing block, requires forging all its successor blocks
Blockchain

Bitcoin’s Consensus

- How to decided which block should be included in the Blockchain?
  - Consensus!
- Consensus can be solved with $f \leq \frac{n}{2}$ (as in bitcoin) only if:
  - The model uses signed messages
  - The network is synchronised
- Bitcoin uses a leader based consensus
  - Every round has a unique leader that publishes its block
  - As the nodes are not pre-known, how do we choose a leader?
Blockchain
Mining and PoW

- Each node tries to solve a puzzle. The one that succeed, is the leader
  - The puzzle is to find a partial collision in a hash
  - Believed to be an NP-hard problem that can be solve only by brute force
  - The puzzle difficulty is adjusted to the network latency such that with high probability, the puzzle will be solved only once in a round
- What if accidentally two different node were able to solve the puzzle?
  - Bitcoin allows ”forks” in the chain
  - A node always prefers the longest version it knows about
  - This approach also handles failures (how?)
  - What are the main drawbacks of this approach?
- To forge a new version of length $n$ a node must solve $n$ PoWs
  - As long 51% of the network’s CPU power are honest, the honest chain will always be the longest
- Solving the PoW is called **mining**
  - Mining requires a lot of CPU power. Hence the miner is rewarded with 25 bitcoins
Bitcoin in Practice
Transactions

Overview

- A transfer of Bitcoin value that is broadcast to the network and collected into blocks
- Once transactions are buried under enough confirmations they can be considered irreversible
- Transactions are not encrypted, so it is possible to browse and view every transaction ever collected into a block
- Previous transaction outputs serves as new transaction inputs
  - An unordered transactions for which we have input transactions are gathered in the **Transactions Pool**
  - Transaction that is missing one or more input transactions is called an **orphan transaction** and entered to the **Orphan Transactions Pool** instead of the transactions pool
Transactions

Propagating Transactions

- A new validated transaction injected into any node on the network will be sent to all of the nodes connected to it (neighbors), each of which will send the transaction to all its neighbors, and so on.
- To prevent spamming, denial-of-service attacks, or other nuisance attacks against the bitcoin system, every node independently validates every transaction before propagating it further.
  - A malformed transaction will not get beyond one node.
Transactions

Transaction pools

- Nodes use the Transactions pool to keep track of transactions that are known and verified to the network but are not yet included in the blockchain.
- When a transaction is added to the transaction pool, the orphan pool is checked for any orphans that reference this transaction’s outputs. Any matching orphans are then validated. If valid, they are removed from the orphan pool and added to the memory pool.
- When a new block is received, all transactions that conflicts with it are removed from the pool.
Blocks

Overview

- Transaction data is permanently recorded in files called blocks
- Blocks are organized into a linear sequence over time (Blockchain)
- New transactions are constantly being processed by miners into new blocks which are added to the end of the chain
- As blocks are buried deeper and deeper into the blockchain they become harder and harder to change or remove
## Blocks

### Block Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic no</td>
<td>value always 0xD9B4BEF9</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Blocksize</td>
<td>block size (Bytes)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Blockheader</td>
<td>consists of 6 items</td>
<td>80 bytes</td>
</tr>
<tr>
<td>Transaction counter</td>
<td>positive integer</td>
<td>1 - 9 bytes</td>
</tr>
<tr>
<td>transactions</td>
<td>list of transactions</td>
<td>#Transactions</td>
</tr>
<tr>
<td>Field</td>
<td>Purpose</td>
<td>Updated when</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>version</td>
<td>Block version number</td>
<td>On upgrading the software</td>
</tr>
<tr>
<td>hashPrevBlock</td>
<td>hash of the previous block header</td>
<td>A new block arrival</td>
</tr>
<tr>
<td>hashMerkleRoot</td>
<td>256-bit hash based on all of the transactions in the block</td>
<td>A transaction is accepted</td>
</tr>
<tr>
<td>Time</td>
<td>Current timestamp as seconds since 1970-01-01T00:00 UTC</td>
<td>Every few seconds</td>
</tr>
<tr>
<td>Bits</td>
<td>Current target in compact format</td>
<td>The difficulty is adjusted</td>
</tr>
<tr>
<td>Nonce</td>
<td>32-bit number (starts at 0)</td>
<td>A hash is tried (increments)</td>
</tr>
</tbody>
</table>
Block Header
PoW

- A block $b$ is valid only if $h(b) < \text{Bits}$
- $h(b)$ is depended on $\text{Nonce}$
- Solving the puzzle is achieved by repeatedly increment $\text{Nonce}$ until $h(b) < \text{Bits}$
Block Header

Merkle Tree

- Used to verify that a transaction is included in a block
- $2 \log(n)$ nodes needed to be sent
Bitcoin’s Network

Overview

- The collection of nodes running the bitcoin P2P protocol
  - maintain a complete and up-to-date copy of the bitcoin blockchain
  - relies on the network to receive updates about new blocks of transactions, which it then verifies and incorporates into its local copy of the blockchain
- The entire network forms a loosely connected mesh without a fixed topology or any structure
- Messages, including transactions and blocks, are propagated from each node to all the peers to which it is connected - flooding
- A node picks several (10 or more) random known nodes and connects to them as his output nodes
- A node may accept any number of input nodes
- An output contact is considered unavailable after a timeout (90 minutes by default), which triggers contacting another random node
Bitcoin’s Network

Network Discovery

- Bitcoin network topology is not geographically defined
- A new node must discover at least one existing node on the network and connect to it
- How does a new node find peers?
  - DNS seeds - provide a list of IP addresses of bitcoin nodes
  - Already known seed node
Bitcoin’s Network

Network Discovery

- To connect to a known peer, nodes run a handshake protocol
  - A new node sends (to an already existing node) a **version** message which contains basic identifying information
  - The receiver responds with **verack** to acknowledge and establish a connection
  - It then sends its own version message if it wishes to reciprocate the connection and connect back as a peer
  - The nodes sets the version to the minimum among the two
# Bitcoin’s Network

## Version Message (main fields)

<table>
<thead>
<tr>
<th>Field Size</th>
<th>Description</th>
<th>Data type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>version</td>
<td>int32_t</td>
<td>Identifies protocol version being used by the node</td>
</tr>
<tr>
<td>8</td>
<td>services</td>
<td>uint64_t</td>
<td>bitfield of features to be enabled for this connection</td>
</tr>
<tr>
<td>8</td>
<td>timestamp</td>
<td>int64_t</td>
<td>standard UNIX timestamp in seconds</td>
</tr>
<tr>
<td>26</td>
<td>addr_recv</td>
<td>net_addr</td>
<td>The network address of the node receiving this message</td>
</tr>
<tr>
<td>6</td>
<td>addr_from</td>
<td>net_addr</td>
<td>The network address of the node emitting this message</td>
</tr>
<tr>
<td>4</td>
<td>start_height</td>
<td>int32_t</td>
<td>The last block received by the emitting node</td>
</tr>
</tbody>
</table>
Bitcoin’s Network
Address propagation and discovery

- Once one or more connections are established, the new node will send an `addr` message containing its own IP address to its neighbors.
- The neighbors will forward the `addr` message to their neighbors, ensuring that the newly connected node becomes well known and better connected.
- Additionally, the newly connected node can send `getaddr` to the neighbors, asking them to return a list of IP addresses of other peers.
Bitcoin’s Network
Exchanging Inventory

- The first thing a full node will do once it connects to peers is try to construct a complete blockchain.
- The process of syncing the Blockchain starts with the version message that contains `start_height`.
- Peered nodes will exchange the `getblocks` message that contains the hash of the top block on their local blockchain. That is to locate who’s got the longest blockchain.
- That peer will identify the first 500 blocks to share and transmit their hashes using an `inv` (inventory) message.
- The node missing these blocks will then retrieve them, by issuing a series of `getdata` messages requesting the full block data and identifying the requested blocks using the hashes from the `inv` message.
- The above method is actually a `pool-gossip` procedure.
Bitcoin’s Network
Exchanging Inventory

- The node keeps track of how many blocks are "in transit" per peer connection, checking that it does not exceed a limit.
- This way, if it needs a lot of blocks, it will only request new ones as previous requests are fulfilled, allowing the peers to control the pace of updates and not overwhelming the network.
Bitcoin’s Network
Exchanging Inventory

Node A

getblocks
getblocks

inv
getdata

block
block
block
block
block
block
block

Node B

TIME

Yehontan Buchnik (Based on Dolev Adas slides)  Distributed Systems 236351  December 20, 2018, 23 / 29
Bitcoin’s Nodes

Overview

- A node that runs the full protocol is called a full node
- But, most of the users are not interested in mining
  - These users can run Simplified Payment Verification (SPV) Nodes
Bitcoin’s Nodes

SPV Node

- Not all nodes have the ability to store the full blockchain, such as smartphones, tablets, or embedded systems
- SPV nodes download only the block headers and do not download the transactions included in each block
- Relies on peers to provide partial views of relevant parts of the blockchain on demand to verify transactions
- SPV nodes use a `getheaders` message instead of `getblocks`
- The responding peer will send up to 2,000 block headers using a single headers message
SPV Node

Hidden Transactions

- An SPV node cannot be persuaded that a transaction exists in a block when the transaction does not in fact exist.
- The SPV node establishes the existence of a transaction in a block by requesting a merkle path proof and by validating the proof of work in the chain of blocks.
- Relies on peers to provide partial views of relevant parts of the blockchain on demand to verify transactions.
- However, a transaction’s existence can be "hidden" from an SPV node:
  - Cannot be sure that a coin did not already spent (the double spending problem).
  - Hence, an SPV node needs to connect randomly to several nodes, to increase the probability that it is in contact with at least one honest node.
Because SPV nodes need to retrieve specific transactions in order to selectively verify them, they also create a privacy risk.

Unlike full blockchain nodes, which collect all transactions within each block, the SPV node’s requests for specific data can inadvertently reveal the addresses in their wallet.

Shortly after the introduction of SPV/lightweight nodes, the bitcoin developers added a feature called bloom filters to address the privacy risks of SPV nodes.
Bloom Filters

Overview

- Space-efficient probabilistic data structure
- Test whether an element is a member of a set
- Bloom filters are implemented as a variable-size array of $M$ binary bits and $K$ different independent hash functions
- False positive matches are possible, but false negatives are not
SPV Node
Using Bloop Filter

- An SPV node initializes a bloom filter as empty (the bloom filter does not match any patterns)

- Then the SPV node initializes the bloom filter with the transaction it’s care about and sends a **filterload** message to the peer, containing the bloom filter to use on the connection

- The receiver replays with all the transactions that matches the filter