Bitcoin and Nakamoto Consensus

Distributed Systems

Nikita Borisov
Topics for Today

• Bitcoin
  • Consensus approach
  • Transaction broadcast
Bitcoin

• Implement a distributed, replicated state machine that maintains an *account ledger (= bank)*
• Scale to thousands of replicas distributed across the world
• Allow old replicas to fail, new replicas to join seamlessly
• Withstand various types of attacks
Approaches that don’t work

• Totally ordered multicast (e.g., ISIS)
  • Quadratic communication overhead
  • Do not know who all replicas are a priori

• Leader election (e.g., Bully)
  • Quadratic communication overhead
  • Do not know who all replicas are a priori
  • *Nodes with highest IDs are leaders =>*
    • Bottleneck
    • Security
Lottery Leader Election

• Every node chooses a random number
• Leader = closest to 0
Hash Functions

• Cryptographic hash function: \( H(x) \rightarrow \{ 0, 1, \ldots, 2^{256}-1 \} \)

• Hard to *invert*:
  - Given \( y \), find \( x \) such that \( H(x) = y \)

• E.g., SHA256, SHA3, ...

• Every node picks random number \( x \) and computes \( H(x) \)
• Node with \( H(x) \) closest to 0 wins
Using a seed

• Every node picks x, computes H(seed || x)
  • Closest to 0 wins

What to use as a seed?

• Hash of:
  • Previous log
  • Node identifier
  • New messages to add to log

• Two remaining problems:
  • How to find closest to 0?
  • How to prevent nodes from trying multiple random numbers?
Iterated Hashing / Proof of work

- Repeat:
  - Pick random x, compute $y = H(seed || x)$
  - If $y < T$, you win!

- Set threshold $T$ so that on average, one winner every few minutes

- E.g.:
  - 1000 nodes
  - $10^{12}$ hash/second
  - Target interval: 10 minutes
  - $T = ?$

- Given a solution, $x$ such that $H(seed || x) < T$, anyone can verify the solution in constant time (microseconds)
Bitcoin Summary

To commit a transaction to the log, a server must solve a puzzle:

$$H(\text{previous log} \mid \text{new tx} \mid \text{random input}) < T$$

Any random input satisfies this with probability of $T/2^{256}$ so smaller $T$ => more attempts needed to solve puzzle

Any node can verify the puzzle solution in one step
Block

Block B1

Log entries
...
...
Puzzle solution

H(B1)
= H(log entries \| solution) < T

Block B2

H(B1)

Log entries
...
...
Puzzle solution

Append Entries (e_1, e_2, e_3, ..., e_k)

H(B2)
= H(B1) || B2
Bitcoin Node Operations

1. Clients broadcast transactions, servers collect unlogged transactions into mempool
2. Servers create candidate blocks, try to solve puzzle
3. If puzzle solved, add to log & broadcast block
4. If receive broadcast, verify solution & add to log
5. Restart step 2 with new log
Consensus Rules

A broadcast block is accepted into the log if it satisfies consensus rules:

- Puzzle solution correct
- Block contents well-formed
- Transactions are authorized (digital signature of account owner)
- Balances don’t go below 0

$\text{Account} = \text{public key}$

$\begin{align*}
K_{\text{pub}} & \quad & K_{\text{sec}} \\
K_{\text{pub}} & \quad & K_{\text{sec}} \\
\end{align*}$

\[\text{auth}(M, K_{\text{sec}}) \Rightarrow \text{sig}(M, sig, K_{\text{pub}}) = Y\]

$\text{UTXO} - \text{unspent transaction output}$
Double-spending

Account balance: $1000
TX1: Pay landlord $800
TX2: Buy TV for $750

At most one of these is valid
Forks

Two different blocks extending a single log == fork

Server solves puzzle before hearing another solution
• Reduce chance by making block time large, broadcast efficient

Server *chooses* to use old log to rearrange transactions
• Longest chain rule, incentives
Chaining

• Each block’s puzzle depends on the previous one
  • \( L_n \rightarrow L_{n-1} \rightarrow \ldots \rightarrow L_1 \rightarrow L_0 \)
  • To add \( m \) blocks, must solve \( m \) puzzles

• Longest chain wins
Chain evolution

Alice

Bob

Charlie

David

2019-03-12

Nikita Borisov - UIUC
Incentives for Logging

• Security better if more people participated in logging
• Incentivize users to log others’ transactions
  • Transaction fees: pay me x% to log your data
  • Mining reward: each block creates bitcoins
    • Replace “Alice minted x” entries with “Alice logged line L_n”
• Payment protocol:
  • Alice->Bob: here’s coin x
  • Broadcast to everyone: Alice transfers x to Bob
  • Bob: wait until transfer appears in a new log line
    • Optionally wait until a few more lines follow it
Putting it all together

Alice generated 50 BTC
Nonce: 1234

Bob generated 50 BTC
Nonce: 5678

Carol generated 50 BTC
Alice transferred 10 BTC to Bob + 1 BTC to Carol (fee)
Nonce: 9932

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>39 BTC</td>
</tr>
<tr>
<td>Bob</td>
<td>60 BTC</td>
</tr>
<tr>
<td>Carol</td>
<td>51 BTC</td>
</tr>
</tbody>
</table>
Bitcoin Recap

• Clients broadcast transactions
  • Transfers between accounts, authenticated using public key signatures

• Miners create candidate blocks of transactions

• Miners solve puzzles to append blocks to log
  • Hard to solve, easy to verify

• Forks can happen, but ...
  • Unlikely if broadcast speed << logging speed
  • Blocks / puzzles are chained, longest chain rule resolves forks quickly

• Mining incentivized by fees, block reward
  • Many miners => harder to create alternate history & double-spend

\[
def \# \text{ of confirmations of tx} = \# \text{ of blocks after tx has been logged}\]
Logging Speed

• How to set T?
  • Too short: wasted effort due to broadcast delays & chain splits
  • Too long: slows down transactions

• Periodically adjust difficulty $T$ such that one block gets added every 10 minutes
  • Determined algorithmically based on timestamps of previous log entries

• Current difficulty
  • $9 \times 10^{22} \approx 2^{76}$ hashes to win

• Large number of participants: hard to revise history!
Bitcoin broadcast

• Need to broadcast:
  • Transactions to all nodes, so they can be included in a block
  • New blocks to all nodes, so that they can switch to longest chain

• Why not R-multicast?
  • Have to send $O(N)$ messages
  • Have to know which nodes to send to
Gossip / Viral propagation

• Each node connects to a small set of neighbors
  • 10–100

• Nodes propagate transactions and blocks to neighbors

• Push method: when you hear a new tx/block, resend them to all (some) of your neighbors (flooding)
• Pull method: periodically poll neighbors for list of blocks/tx’s, then request any you are missing
Push propagation
Pull propagation

What transactions do you know?

Node 1: Tx1, tx7, tx13, tx25, tx28

Node 2: Please send me tx13, tx28

Contents of tx13, tx28
Maintaining Neighbors

• A *seed* service
  • Gives out a list of random or well-connected nodes
  • E.g., seed.bitnodes.io

• Neighbor discovery
  • Ask neighbors about *their* neighbors
  • Randomly connect to some of them
Bitcoin summary

Foundations:
• Unreliable broadcast using gossip
• Probabilistic “leader” election for mining blocks (tx ordering)
• Longest chain rule to ensure long-term consistency / security

Compared with Paxos/Raft:
• Scales to thousands of participants, dynamic groups
• Tens of minutes to successfully log a transaction (vs. milliseconds)
Other Bitcoin Topics

• Mining pools to distribute winnings
• Security analysis, selfish-mining strategies, “incentive compatibility”
• Network design, gossip improvements, logging speed
• Block size, block propagation, block compression
• Private versions: Monero, Zcash, ...

• Ethereum: build a full-scale virtual machine
  • Instead of executing transactions, execute programs
  • Programs maintain/update state, transfer money
  • Smart contracts control an account, receive commands