Braiding Techniques to Improve Security and Scaling

https://github.com/Taek42/jute

About Me

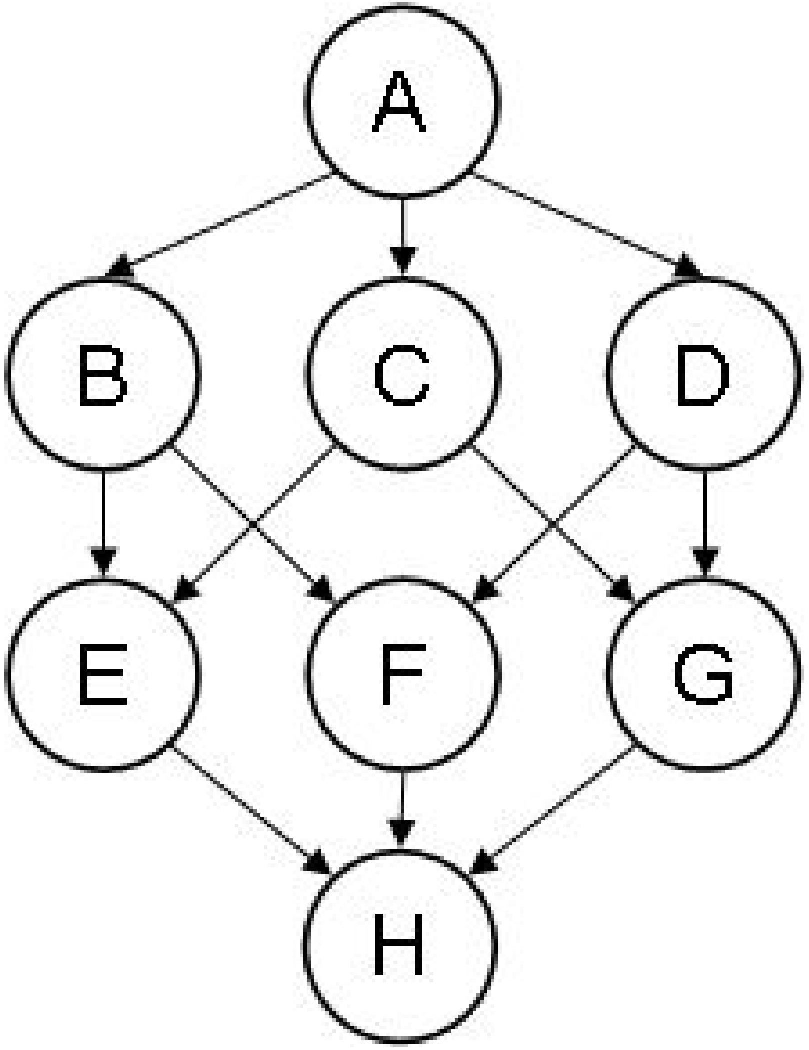
- David Vorick
- Bitcoin since 2011
- Full-time blockchain engineer since 2014

- CEO of Nebulous Inc.
- Lead developer for Sia - functional decentralized cloud storage

- https://sia.tech

What is Braiding?

- Blocks can have multiple parents
- Consensus achieved by a combination of sorting and deleting
- ‘Inclusive’ braiding means conflicting blocks are allowed to coexist



Why Braid? / Shortcomings with Satoshi Consensus

- Low latency, high hashrate have more profits due to lower orphan rates
- Large miners can engage in selfish mining.
- Network is idle for 10 minutes at a time, then busy for a few seconds.
- Confirmations can take more than 30 minutes.
- Only 4,000 payouts per month, solo-mining impractical without millions of dollars. Mining pools offer a solution, but are a centralization pressure.

Non Braiding Solutions

- Can reduce the symptoms
- Can’t eliminate the problems entirely
- Often fail during adversarial conditions
- Nothing addresses selfish mining

- The Relay Network, FIBRE
- IBLT
- Weak Blocks
- Compact Blocks
- Thin Blocks
- P2Pool
- (other solutions as well)

Existing Braid Work

- Often Complex
- Often Confused Security Models
- Often vulnerable to strategic mining
- Often easy to cause deep reorgs (need < 51% hashrate)

- Bob McElrath's Braids
  https://github.com/mcelrath/braidcoin
- GHOST
  https://eprint.iacr.org/2013/881.pdf
- Inclusive Blockchain Protocols
  http://fc15.ifca.ai/preproceedings/paper_101.pdf

(and more)

Jute: A New Approach to Braids

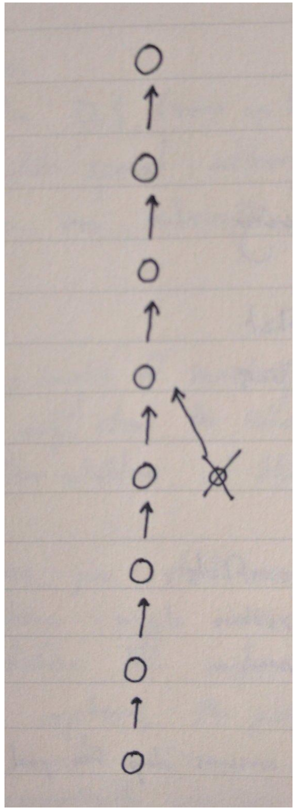
+ Easy to understand
+ More thorough security model
+ Reasonable computation + engineering demands
+ Working Code

+ ~6 second block times
+ ~50kb block sizes
+ **No orphans, no selfish mining**
+ Need 50+ confirmations for security - 5 minutes (+/- a minute)
+ ~400,000 blocks per month (good for solo mining)

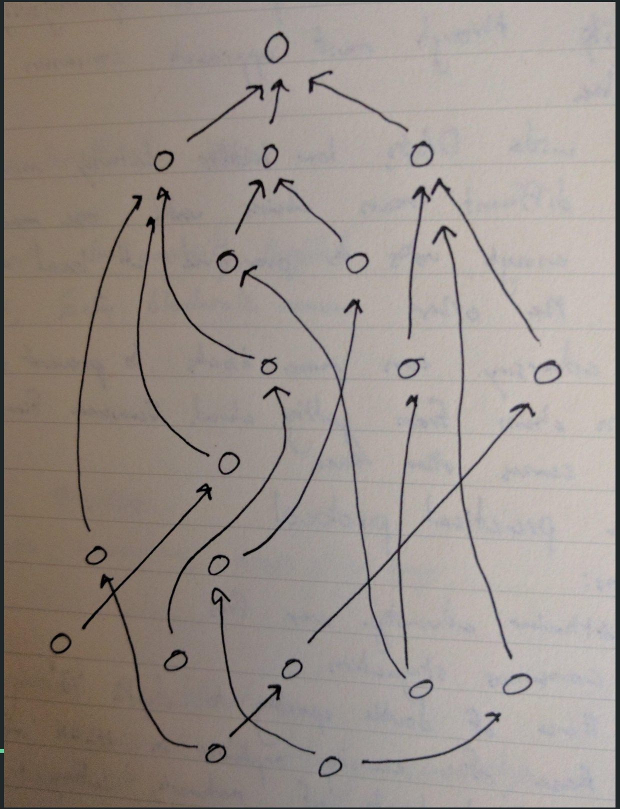
Graph Topology for Bitcoin + Jute

- Bitcoin has a 10 minute block time. At any given point in time, most/all nodes will have seen all blocks that have been found. Sometimes 1 or 2 blocks have been found but have not yet propagated to all nodes.
- In Jute, most of the time there is at least one block which has been found, yet hasn’t propagated. Sometimes, 6+ blocks will have been found but not yet propagated.
- Low latency miners in Jute are far more likely to have seen most/all recent blocks than high latency miners. A low latency may on average know about every block, while a high latency miner on average may not know about the 2-4 most recent blocks.

Bitcoin



Jute



Requirements of a Competitor to Satoshi Consensus

+ Immutable history. Once a block is ‘confirmed’, self-interested miners will continue confirming that block such that it gets harder and harder to rewrite history as deep as that block.
+ Miners with less than 51% hashrate should not have any incentives to be break protocol - any alternative behavior should negatively affect profits.
+ Miners with less than 51% hashrate should not be able to censor transactions from the network, nor should they be able to rewrite confirmed history.
+ SPV verification needs to be supported
+ Miners should not gain advantages by incrementally increasing their hashpower - there should be minimal centralization pressure
+ Miners should not gain advantages through strategic behavior or improved network connectivity.

Surface Area Included in Requirements

+ Denial-of-Service attacks - processing the chain, propagating blocks, performing reorganizations all need to be DoS free. Worst-case must be tolerated by low-power nodes.
+ Transaction fee manipulations - users can create conflicting transactions, miners can mine conflicting transactions. With low enough block times, low latency miners can steal fees from high latency miners (be re-mining and propagating)
+ Double spend attempts - how hard is it for a high hashrate miner to execute a double spend? How expensive is it to fail?

Managing the Chaos of Jute

- As much as possible, Jute wants to be inclusive, is it is fair to high latency, low hashrate miners. All blocks, even those with conflicting transactions are included.
- Invalid transactions are thrown out while processing the chain, however the blocks that included these transactions are kept, and the other transactions in the block are also kept.

Foundation for an Immutable Sorting

+ Consensus in jute is built by picking out a main chain from the block graph, and then iteratively merging all ancestors of each node in the main chain.
+ The main chain is determined by votes. Votes are applied toward edges. When extending the graph, you start from the genesis block and iteratively find the child whose edge has the most votes. Then you add a vote to that edge. Eventually, the new block will be reached, at which point voting is complete.
+ Two things skipped - tie breaks, and a censorship protection. We will come back to these.

Simple Example

$$\begin{tikzcd}
& 0 & \\
1 \arrow[ur, "1"] & & 2 \arrow[ul, "1"']
\end{tikzcd}$$

$$\begin{tikzcd}
& 0 & \\
1 \arrow[ur, "2"] & & 2 \arrow[ul, "1"'] \\
& 3 \arrow[ul, "1"] \arrow[ur, "0"'] &
\end{tikzcd}$$

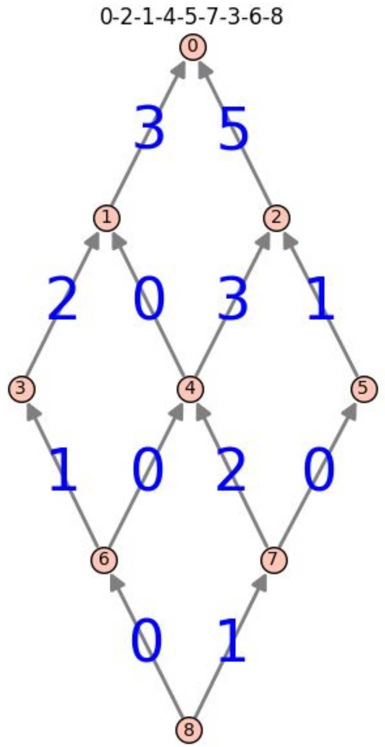
$$\begin{tikzcd}
& 0 & \\
1 \arrow[ur, "5"] & & 2 \arrow[ul, "1"'] \\
& 3 \arrow[ul, "4"] \arrow[ur, "0"'] & \\
& 4 \arrow[u, "3"] & \\
& 5 \arrow[u, "2"] & \\
& 6 \arrow[u, "1"] &
\end{tikzcd}$$

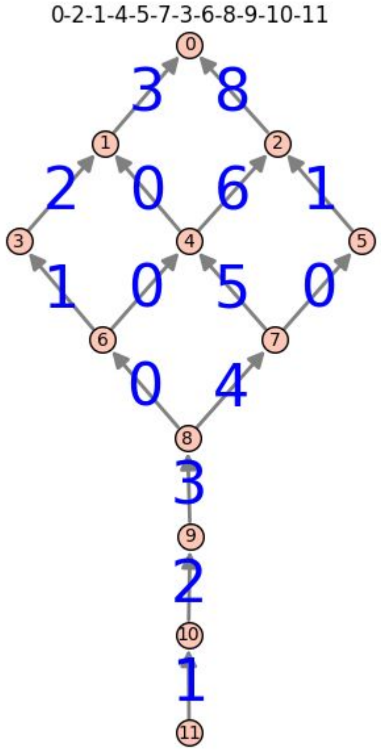
Tie Breaks

Sometimes, multiple children of a block in the main chain will have the same number of votes. To choose between them if there's a tie:

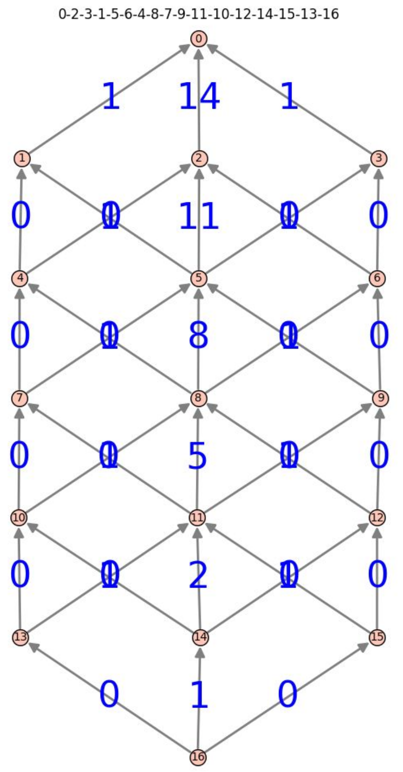
1. Favor the one with the most ancestors
2. If still tied, use hash of merging block as RNG seed, pick one randomly

Nested Diamond Example





No Collapse Example



Reorganizing the Chain

+ To pick a new main chain, you need to upgrade an edge to having more moves
+ Every block that is produced confirms the existing main chain, these edges are continuously getting stronger, while no votes are going to the competing edges at all
+ An attacker will need to add strength to their alternate main chain faster than the rest of the miners are strengthening the main chain
+ Need >51% hashrate. The main chain is protected with the same strength as Bitcoin. *almost

*almost ?

+ The network cannot confirm blocks in the main chain if those blocks have not yet propagated
+ Each node will have a different view of the network, they may see different main chains
+ Convergence depends on a majority vote for a particular edge propagating to all supporters of its competitors before those competitors can get extra votes
+ When block time is too far below the network propagation time, convergence is not guaranteed, and can even be actively disrupted by a high hashrate attacker.
+ Unclear if this is a solvable problem. Solutions desired!

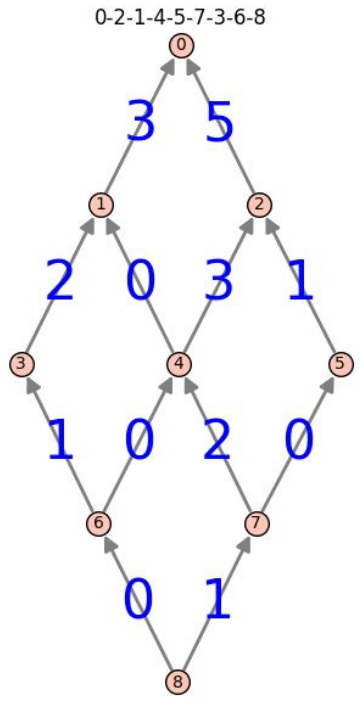
Ensuring Convergence

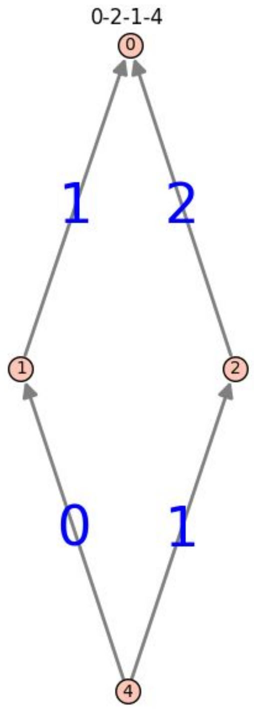
- This is the primary reason that the block time is set to 6 seconds, and not lower.
- The network propagation time for 50kb blocks is estimated to be below 12 seconds for the majority of hashrate.
- Blocks are found at a highly variant rate, if there are two competing main chains, one should win by luck in under a minute
- This is handwavy, not properly tested. Maybe 6 seconds is too ambitious.

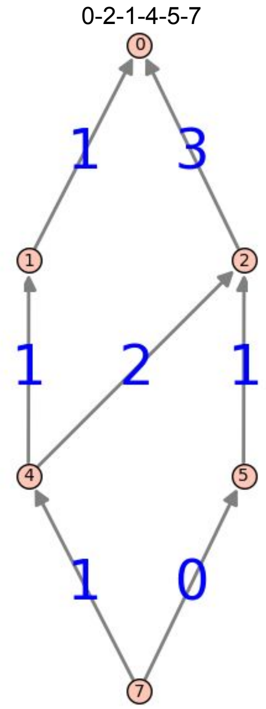
Sorting Off-Main-Chain Blocks

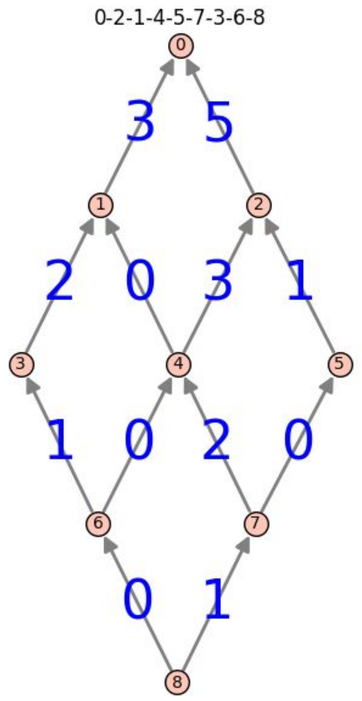
- No parent block should be sorted after its child
- History should not change unless the main chain changes
- For each block in the main chain (starting from Genesis), first include all ancestors that have not already been included. Then include the block, and move to the next block in the main chain.
- To guarantee that history cannot change without changing the main chain, sort parents by viewing the graph as the child block saw the graph.
- Sort by finding a ‘sub’ main-chain in the off-chain blocks

Sorting Example









Code

https://github.com/Taek42/jute

+ The ‘consensus-poc’ folder contains code which can assemble graphs, add the votes according to the rules described, and then sort blocks according to the rules described
+ Check out the README
+ ‘NewGraph()’ to make a graph using the rules described so far
+ ‘NewLowBlockTimeGraph()’ to make a graph using the extra rule that I haven’t described yet (irrelevant for most graphs, irrelevant when block time is high)

Revisiting Problems (solutions in upcoming slides)

+ Transaction fees work entirely differently. Optimal mining strategies change, potential attack vectors change, it’s a new ecosystem. All (known) potential problems have solutions, but it’s a big change from mining on a Satoshi consensus blockchain.
+ If you allow blocks to be included from infinitely back, a medium hashrate attacker can spend months creating blocks and then dump them all on the network at the same time, a DoS attack.
+ No orphans and no high-latency penalties means that double spend attempts are essentially free. Need additional protection.

Non-Adversarial Transaction Fees

+ Low latency miners are going to be able to usually get their blocks into the chain ahead of high latency miners, which means they get preference.
+ Low latency miners have further advantage because they will see sooner which fees have been mined in a block, and know faster not to mine those fees.
+ It's bad for everyone if multiple blocks are mined with high transaction overlap. At least one of the miners loses out on fees, effective network throughput is lower.
+ Miners therefore want to mine probabilistically, adding transactions randomly weighted by how big the fee is
+ Most effective when there are a lot of potential transactions with similar fees. Deep mempool also limits the advantages of low latency miners.

Incentivizing a Deep Mempool

- Make it expensive to have blocks with more transactions in them. Require miners to burn coins for every transaction they add, increasing as blocks grow.
- E.g., need to burn 1 satoshis per byte for first kilobyte, 2 satoshis per byte for second kilobyte, etc. (blocks are 50kb in jute, so max burn for a whole block is 25 satoshis per byte)
- If there are not enough transactions to fill blocks all the way up, transaction fees will be lower, low enough that miners will prefer to burn less coins and spread the transactions out, effectively making the transaction pool deeper.

Redistributing Fees is not Byzantine Fault Tolerant

+ Any solution that redistributes fees, such as fee splitting or a mining fee pool, is a bad idea.
+ This is because it means the effective fee on a transaction goes down, as that fee is at risk of being redistributed
+ Transaction creators then have incentives to use alternate mechanisms to pay miners for mining transactions, such as making a special hidden output or doing other backroom / out-of-band deals
+ Backroom deals are more accessible to people with better social connections, and are therefore a centralization pressure.
+ Fees must be paid in full to the miner than mined the transaction, if and only if that transaction is determined to be valid by consensus.

Adversarial Transaction Fees

- Miners can intentionally launch a bunch of double spends in an attempt to hurt their competitors. Low latency miners will be hurt less than this by high latency miners (as they know faster which transactions to not mine), which is a form of selfish mining
- With a low enough block time (6 seconds probably not low enough), adversaries can re-mine blocks from high-latency miners, stealing all the fees. When you see a block from a slow miner, you re-mine it, giving all the fees to yourself, and then you propagate it fast enough that it beats the slow miner.

Fixing the Adversarial Cases

- To not be a victim to potential double spends, wait a random number of seconds (0 - 12 seconds) before mining a transaction. If there is a double spend, it'll propagate and you'll know to not mine it.
- Fee re-mining can be resolved by randomly selecting between conflicting transactions during the consensus phase. If conflicting transactions / transaction-trees (due to dependencies) appear within 20 blocks of eachother, instead of honoring the first one selected between them randomly weighted by the size of the transaction-tree in bytes. This means the first 20 confirmations are entirely useless, but also means that low-latency high-hashrate miners lose the ability to do fee re-mining. Not really necessary with 6 second block times, absolutely necessary at 1 second block times.

Addressing DoS Blocks

+ If we start to ignore blocks, honest miners become vulnerable to being orphaned. If we don’t have any limits on mining, there are huge DoS vectors.
+ We will limit blocks - blocks must have a relative height that is within 200 of their actual height (gap height of less than 200) to be considered for the chain. Otherwise, just the header is used, and the work added / votes added is ignored.
+ Children of ignored blocks can still be included in the chain, which is why we need to preserve the header information - to verify the relative height of the children.
+ (pedantic note - ‘height’ used for convenience. Full limit takes into account amount of work instead of raw count of descendants)

How to Cause Orphans, Given DoS Protection

- Attacker can cause orphans by causing a reorg 200+ deep.
- Requires that the attacker refuse to include other blocks, hurting the attacker’s relative height and increasing the attacker’s own risk of being orphaned.
- Requires that the attacker find 200 blocks faster than the rest of the network is able to find 200 blocks. If less than 45% hashrate, essentially impractical. Overwhelmingly impractical at 35% hashrate.
- If the attacker finds 210 blocks by the time the rest of the network finds 200 blocks, the attacker can cause 10 honest blocks to be orphaned.
- Strategic mining in Jute is very ineffective. Need > 40% hashrate, and you can affect honest profits by < 0.1%.

How to DoS, Given DoS Protections

- Mine and withhold as many blocks as possible, and when the relative height of your earliest withheld block approaches 200, dump all blocks on the network all at once, causing congestion.
- 40% hashrate attacker can dump up to 133 blocks at a time, sometimes 150 blocks with enough luck.
- 25% hashrate attacker can dump up to 66 blocks at a time.
- Transaction fee preference on these blocks will be low

- This will cause congestion for a few minutes, potentially make double spending easier. Easily detectable, and countered by doubling the confirmation time during the congestion.

Handling Double Spends

+ Double spending requires exponential luck in the number of confirmations.
+ Attempting to double spend 3 confirmations is trivial, costless. Attempting to brute force an 8-bit private key is also trivial, costless.
+ Jute has a block time of 6 seconds. After 5 minutes, the main chain will have ~40-50 confirmations. After 20 minutes, ~190-200 confirmations. And so on.
+ Low latency miners have an advantage, as they can propagate their double spend chains faster than the honest chain.

Double Spend Simulator

I created a simulation to estimate the probability that an attacker can execute a double spend. Simulation has configurable variables such as attacker hashrate, block time, and network propagation time.

‘doublespendsim’ folder of https://github.com/Taek42/jute

+ For 33% hashrate attacker, wait for 50 confirmations (5 minutes)
+ For 40% hashrate attacker, wait for 150 confirmations (15 minutes)
+ For 45% hashrate attacker, wait for 500 confirmations (50 minutes)
+ For high value / high risk transactions, double the confirmations requirement
+ Check the README, it’s got some cool stuff

Incentives Against Double Spends

- What happens if a bunch of miners are attempting double spends?
- Their blocks will not be in the main chain.
- This means lower preference on ordering, which means less likely to get transaction fees.

- If using fee modifications, double spends out to 20 confirmations are basically entirely free, and you should wait 20 blocks in addition to the numbers listed previously

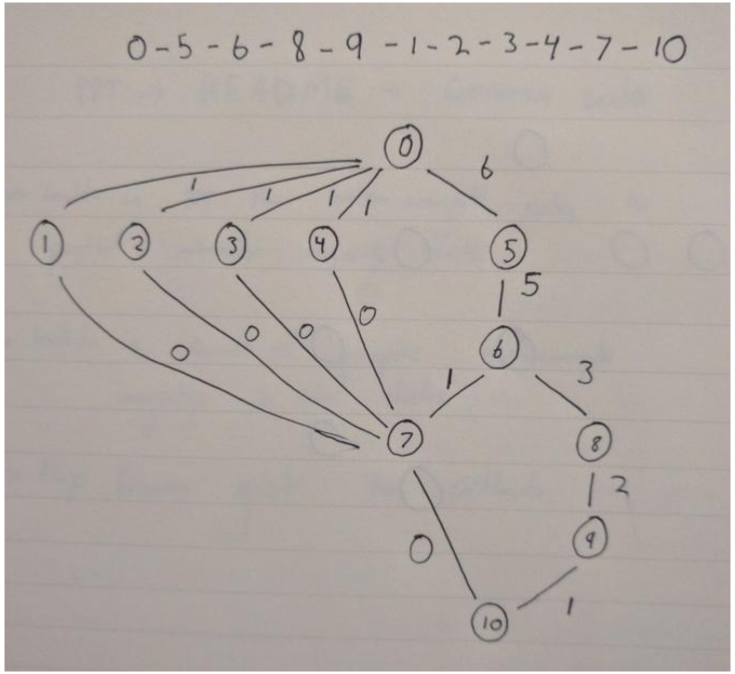
Adding SPV

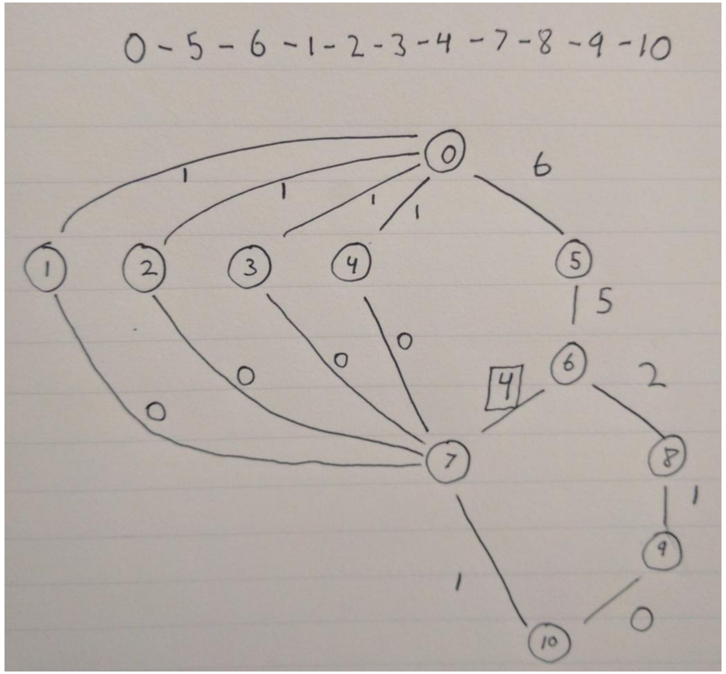
- Transactions will be selectively thrown from blocks, meaning an SPV commitment to the transactions in your own block is not helpful - you need to know which transactions are being thrown away
- Instead of committing to just one block, commit to the valid transactions in the most recent 200 blocks (the maximum gap height). Include the block containing each transaction in the commitment.
- SPV nodes can tell where the reorgs are, and thus know which commitments are potentially invalid.
- SPV nodes have the flexibility of displaying potential transactions to the user and the number of confirmations on those transactions.

Low Block Time Rule

- Only relevant if a minority attacker can consistently mine and propagate multiple blocks per network propagation cycle - meaning block time needs to be very low, and attacker needs to have a significant network advantage
- Much more relevant with insanely low block times, like 100 milliseconds
- Low block time rule not enough to drop the block rate that low safely, because the network-convergence problem is still unsolved.
- **Rule:** If a merging block is introducing blocks to the chain at a higher rate than the main chain is, the edge selected by that block gets extra votes. 'Rate' is determined by number of ancestors visible between main chain and recent common ancestor, vs number of ancestors not visible between main chain recent common ancestor. One extra vote per extra block.
- Precise definition can be found in the code.

Graph Without vs. With Low Block Time Rule





Reiteration of Weaknesses

+ Bitcoin’s strong counter-incentives against double spends get weakened, but double spending is made to require much more luck through waiting for dozens of confirmations.
+ Network is made vulnerable to a limited amount of flooding by an adversary. Flooding is insignificant enough that a permanently ongoing attack should not be disruptive, even from a 49% hashrate miner.
+ Attackers can stall consensus if they can reliably mine multiple blocks per network propagation period. 6 second block time prevents this.
+ New, largely untested. Maybe some more attack vectors exist.

Reiteration of Advantages

+ No more selfish mining attacks
+ No more orphan blocks for high latency miners
+ More frequent blocks means more pipelining + parallelism, more scalability.
+ Reduced emphasis on latency means optimizations can focus instead on throughput.
+ Lower block times mean more practical/accessible solo-mining.
+ Safe number of confirmations in only a few minutes, with low variance. Especially if the largest potential attacker has less than 40% hashrate.

Future Work

+ If we can solve the consensus stalling attack, we can update the protocol to support block times as low as 250 milliseconds, and we can get even stronger scaling benefits, stronger defense against double spends, and you will be able to solo-mine profitably from your bedroom.
+ More research / investigation needs to be done into potential attack vectors
+ Code currently only supports hand-crafted chains, also is grossly unoptimized. We could use an implementation that could be thrown into a testnet of real nodes.
+ Double spend simulator may be inaccurate.

Thanks