Braidpool

Bob McElrath, Ph.D.

TabConf Socratic Village
Oct 15 2022
bob.mcelrath@gmail.com

Braidpool is a (proposal for) a *decentralized mining pool* which:

- Uses a merge-mined DAG with Nakamoto-like consensus to track shares
- Guarantees to pay all miners through consensus rules
- Uses a quorum of miners and a large multi-sig (FROST/ROAST/MuSig2) to sign coinbase payments/settlement transactions
- Tracks owed funds through a “UTXO Set” which is actually a set of transactions (Unspent Hasher Payment Output)
- Targets constant *variance* among participants
- Allows sending of shares which enables hashrate futures and options

Outline

- Shares and Weak Blocks
  - Metadata Commitments
  - Share Value
- Braid Consensus Mechansim
  - Difficulty Retarget Algorithm
  - Miner-Selected Difficulty
  - Simple Sum of Descendant Work
- Eltoo Payout Update
  - Unspent Hasher Payment Output (UHPO)
- Payout Update and Settlement Signing (FROST)

Shares and Weak Blocks

- Braidpool accounts for mining "Shares" (also called “Weak Blocks”)
- A "share" does not meet bitcoin’s difficulty target t_b but meets some lesser difficulty target t. Shares are a bitcoin block header which commits to additional data:

Field	Description
`blockheader`	`Version, Previous Block Hash, Merkle Root, Timestamp, Target, Nonce`
`coinbase`	`Coinbase Txn, Merkle Sibling, Merkle Sibling, ...`
`payout`	`Payout Update Txn, Merkle Sibling, Merkle Sibling, ...`
`metadata`	`Braidpool Metadata` (see below)
`un_metadata`	`Uncommitted Metadata` (see below)

Share Commitments

- A share must commit to the additional data required by Braidpool in the coinbase transaction:

```
OutPoint(Value:0, scriptPubKey OP_RETURN "BP"+<Braidpool Commitment>)
OutPoint(Value:<block reward>, scriptPubKey <P2TR pool_pubkey>)
```

where

`Braidpool Commitment` = hash(`Braidpool Metadata`)

- This prevents third parties from modifying consensus-critical data in a share

Share Metadata

The `Braidpool Metadata` is:

Field	Description
`target`	Miner-selected target difficulty $x_b < x < x_0$
`payout_pubkey`	P2TR pubkey for this miner's payout
`comm_pubkey`	secp256k1 pubkey for encrypted DH communication with this miner
`miner IP`	IP address of this miner
[[`parent` , `timestamp`], ...]	An array of block hashes of parent beads and timestamps when those parents were seen

The `Uncommitted Metadata` block is intentionally not committed to in the PoW mining process. It contains:

Field	Description
`timestamp`	timestamp when this bead was broadcast
`signature`	Signature on the `Uncommitted Metadata` block using the `payout_pubkey`

- The additional timestamps provide required higher resolution bead timing information

Share Value

- Braidpool will use the *Full Proportional* payout method (all rewards and fees distributed proportionally to hashers)
- Braidpool shares have an internal value which is equal to the work w – the estimate of the number of sha256d hashes performed (TeraHash Coin), weighted by the probability that the DAG generated an orphan:

$$P_{\geq 2}(T_C) = \sum_{k=2}^{\infty} \frac{\lambda(T_C)^k e^{-\lambda(T_C)}}{k!} = 1 - e^{-\lambda(T_C)}(1 + \lambda(T_C)) \simeq \frac{\lambda^2(T_C)}{2}$$

$$\lambda(T_C) = \frac{T_C}{\text{block time}} \left(\frac{\text{pool hashrate}}{\text{total hashrate}} \right)$$

$$(\text{share}) \qquad w = \frac{1}{x(1 - P_{\geq 2})}$$

Share Tally

- Shares are a statistical estimate of *work* (number of sha256d calculations performed)
- The BTC value of shares is only known at difficulty adjustment boundaries (every 2016 blocks $\sim$ 2 weeks)
- Shares can only be paid out *after* a difficulty adjustment window closes
- Shares can be sent to a new address (to effect instant payout via atomic swap or exchange trade)

Consensus on a Directed Acyclic Graph (braid)

Allowing blocks to have multiple parents creates a:

Directed Blocks have parents, parents cannot refer to children

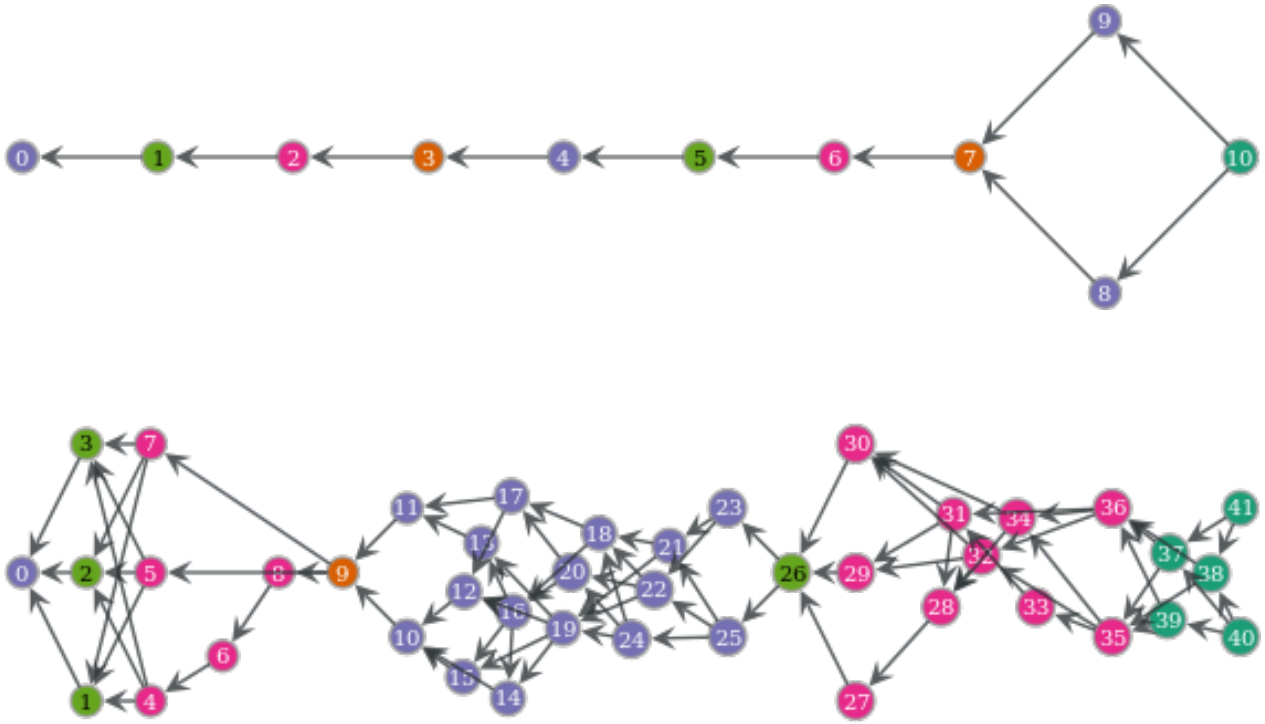
Acyclic A cycle is cryptographically impossible

Graph Structure is non-linear (no “height”)

- A DAG can be *partial ordered* in linear time.
- We have to make a restriction relative to a more general dag, so I’m going to name this data structure a *braid*.

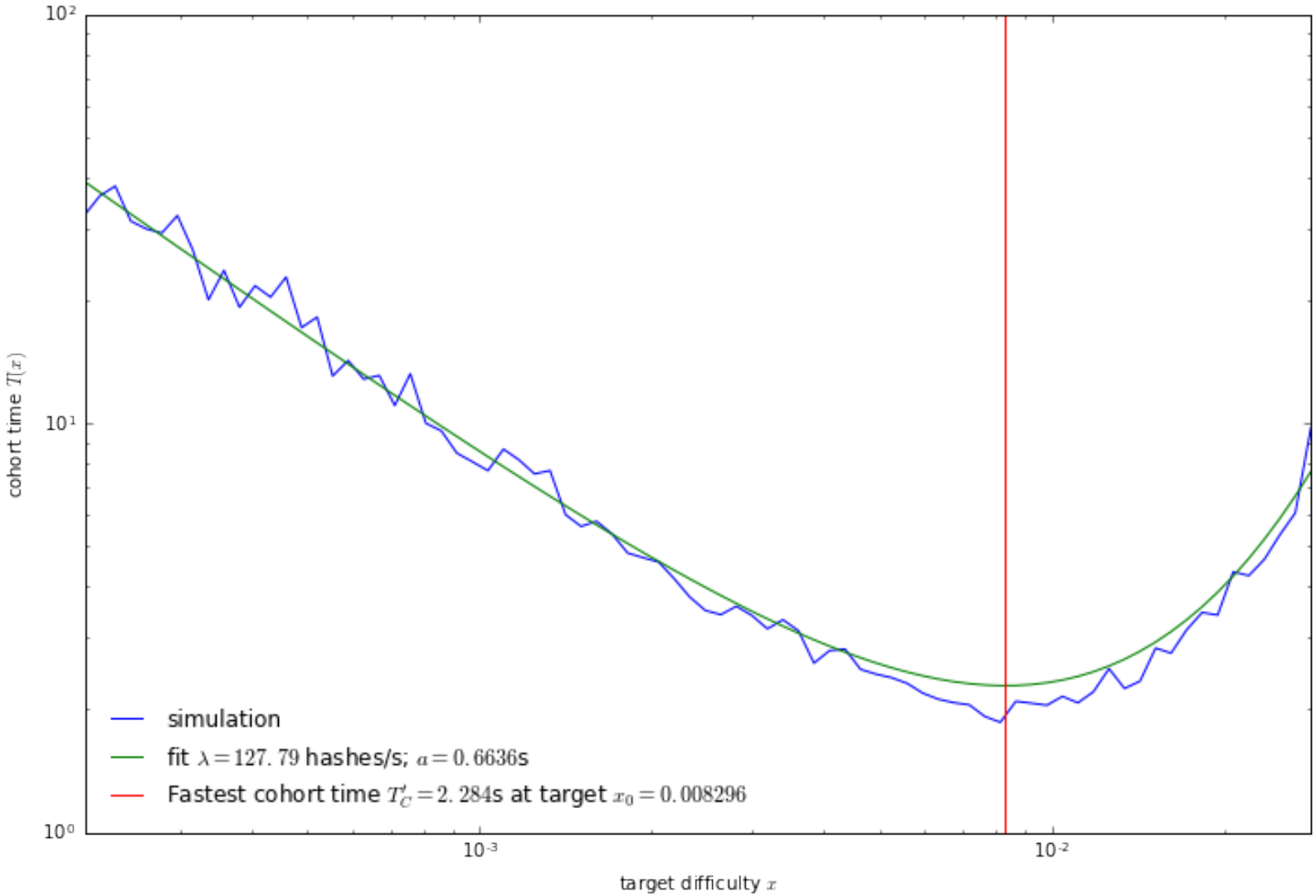
The Cohort

Cohorts define a *total order* such that *all* beads are ancestors of the next cohort, and descendants of the previous cohort:



Cohorts give us a way to measure the latency of the network WRT the bead rate and are used in the retarget algorithm, and in optimizing tx conflict selection.

Cohort Time vs Difficulty



Difficulty Target I

The curve resulting from this simulation is extremely well approximated by

$$T(x) = \frac{1}{\lambda x} + ae^{a\lambda x}$$

- T Cohort time
- λ Hash rate (hashes per second)
- x Target hash value
- a Network "size" parameter (in seconds)

Let's approximate this in the blockchain limit $x \to 0$:

$$T(x) = \frac{1}{\lambda x} + a + \mathcal{O}(x)$$

the quantity a is the *increase in effective block time* due to network latency effects. This is to say that the actual cohort time is slightly longer than expected from the hashrate.

Difficulty Target II

The parameter λ is the hash rate and can be obtained along with a:

$$\lambda = \frac{N_B}{xT_C N_C}; \qquad a = T_C W\left(\frac{T_C}{T_B} - 1\right)$$

where N_B is the number of beads and N_C is the number of cohorts. $W(z)$ is the *Lambert W function*[1]. With these in hand we can compute the location of the minimum

$$x_0 = \frac{2W\left(\frac{1}{2}\right)}{a\lambda} = \frac{0.7035}{a\lambda}$$

This is relatively independent of network topology.

[1] https://en.wikipedia.org/wiki/Lambert_W_function

Miner-Selected Difficulty

- As long as the miners chosen difficulty is $x_b < x < x_0$ there's nothing wrong with allowing miners to choose their own, since we know how to add work.
- Braidpool will choose your difficulty such that all miners have the same *variance*.
- Miners too small to achieve constant variance will be grouped into a *sub-pool*, which:
  - has UHPO outputs in the parent pool as the outputs managed by the sub-pool
  - has an independent Braid from the parent pool

Conflict Resolution: Simple Sum of Descendant Work

For braids having siblings (diamond-like and higher-order structures) where conflict resolution is important will use the Simple Sum of Descendant Work

$$w_{\text{SSDW}} = \sum_{i \in \text{descendants}} w_i = \sum_{i \in \text{descendants}} \frac{1}{x_i(1 - P_{\geq 2}(T_{C,i}))}$$

- Only work within a cohort matters
- In the event of a tie, the smaller hash ("luck") is used.
- We only need conflict resolution if Braidpool has its own transactions.
- Graph structure is manipulable at zero cost, so we ignore it

Payout Update

From the Eltoo paper (Decker, Russell, Osuntokun):

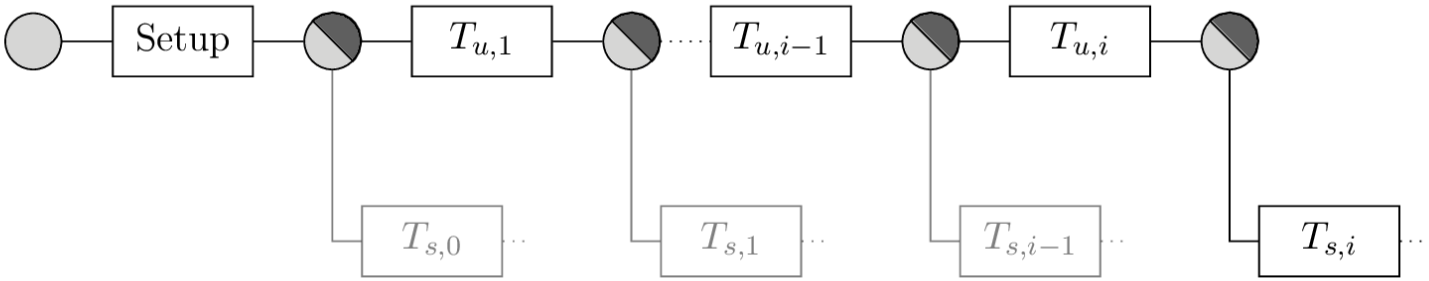


Figure 1: Overview of the on-chain update protocol. The setup transaction initiates the protocol. Each update transaction $T_{u,i}$ invalidates the previously negotiated settlement transaction $T_{s,i-1}$ (indicated by lighter color), until finally $T_{s,i}$ is not invalidated and settles the contract.

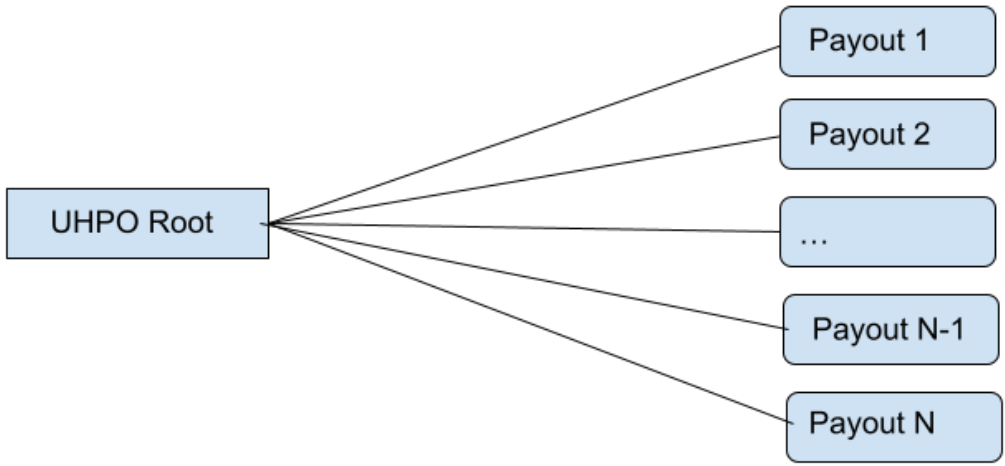
- $T_{u,i}$ takes previous $T_{u,i-1}$ output and a new coinbase as input (optimistic case, single Schnorr signature)
- $T_{s,i}$ is the UHPO transaction tree (spent via tapscript)

Payout Signing

- Each payout update requires two keys: the Update Pubkey (P2TR) and the Settlement Pubkey (in a tapscript).
- These keys are generated by the distributed key generation phase of FROST. All miners who successfully mined past blocks hold FROST keyshares.
- We may combine MuSig2 and FROST and use ROAST as well. (TBD)
- After a bitcoin block is found, a FROST/ROAST signing ceremony is kicked off among signing miners, for both the Update and Settlement transactions, for the most recent unspent Braidpool coinbase at least 100 blocks old.

Settlement Tx: Unspent Hasher Payment Output

- The UHPO “settlement” transaction fans out and pays everyone
- This UHPO transaction is the analog of the “adversarial close” in Lightning
- It is only broadcast in the case of catastrophic failure or shutdown of Braidpool
- Management of this transaction is the analog of the “UTXO Set” for Braidpool.



Derivatives and Instant Payout

- We do not directly create any instant-payout (Lightning-like) method, instead we envision hashers trading shares for BTC
- Professional market makers and risk-taking trading firms would buy shares on centralized exchanges or via atomic swaps. (This is outside the scope of Braidpool, but enabled by it)
- With both shares (per epoch) and BTC as instruments, private hashrate derivative contracts can be created.
- Pools are currently doing this risk management function, and might continue to do so atop braidpool, along with their other services (management, monitoring, financing)

General Timeline

- Braidpool V1
  - Peer-to-Peer Layer (Kulpreet Singh)
  - Braid Consensus (Bob McElrath)
  - Update/UHPO signing (Bob McElrath, Jesse Posner's FROST code from libsecp256k-zkp)
- Braidpool V2
  - Transactions (sending/trading shares)
  - Latency optimization
  - Sub-Pools

Summary/Conclusions

- All the elements are present to construct a truly trustless decentralized mining pool (Shares, Braid consensus, large multi-sigs)
- The advent of Schnorr, FROST, ROAST, and MuSig2 were the missing piece
- This UHPO proposal should be contrasted with the (semi-trusted) Hub model originally proposed by Chris Belcher
- Goal will be to get existing pools to run atop Braidpool, and integrate with Stratum V2