THE MEMPOOL

JOHN NEWBERY

@jfnewbery

github.com/jnewbery

THE MEMPOOL

- What is the mempool
- Mempool limiting and eviction
- Replace-by-fee
- Signature and script caching
- Compact blocks
- Fee estimation

WHAT IS THE MEMPOOL?

WHAT IS THE MEMPOOL?

- A node's view of the set of unconfirmed transactions
- Transactions that are propagated around the network using INVs are stored in nodes' mempools
- Miners select the transactions for the next block from their mempool
- There's no such thing as ***the*** mempool!

MEMPOOL

- Nodes verify transaction validity before accepting the transaction to the mempool
- Nodes also check transactions against *standardness* rules
- Double spends are not allowed in the mempool
- Transaction chains are allowed in the mempool

MEMPOOL LIMITING AND EVICTION

MEMPOOL LIMITING

- A node’s mempool is a private resource
- External parties could potentially abuse that resource
- We need to limit how much resource can be used

MEMPOOL LIMITING AND EVICTION

- Transaction Expiry ages out old transactions (default 14 days)
- Also limiting the mempool size (default 300MB)
- When mempool is full, we evict the lowest-fee paying transactions
- fee-rate calculation is done by ‘package’

FEEFILTER

- A node can tell its peers that it no longer wishes to receive transactions with fees lower than a certain rate
- Peer may stop sending transactions with fee-rate lower than feefilter
- Defined in BIP 133
- Introduced in protocol version 70013

REPLACE-BY-FEE

REPLACE-BY-FEE

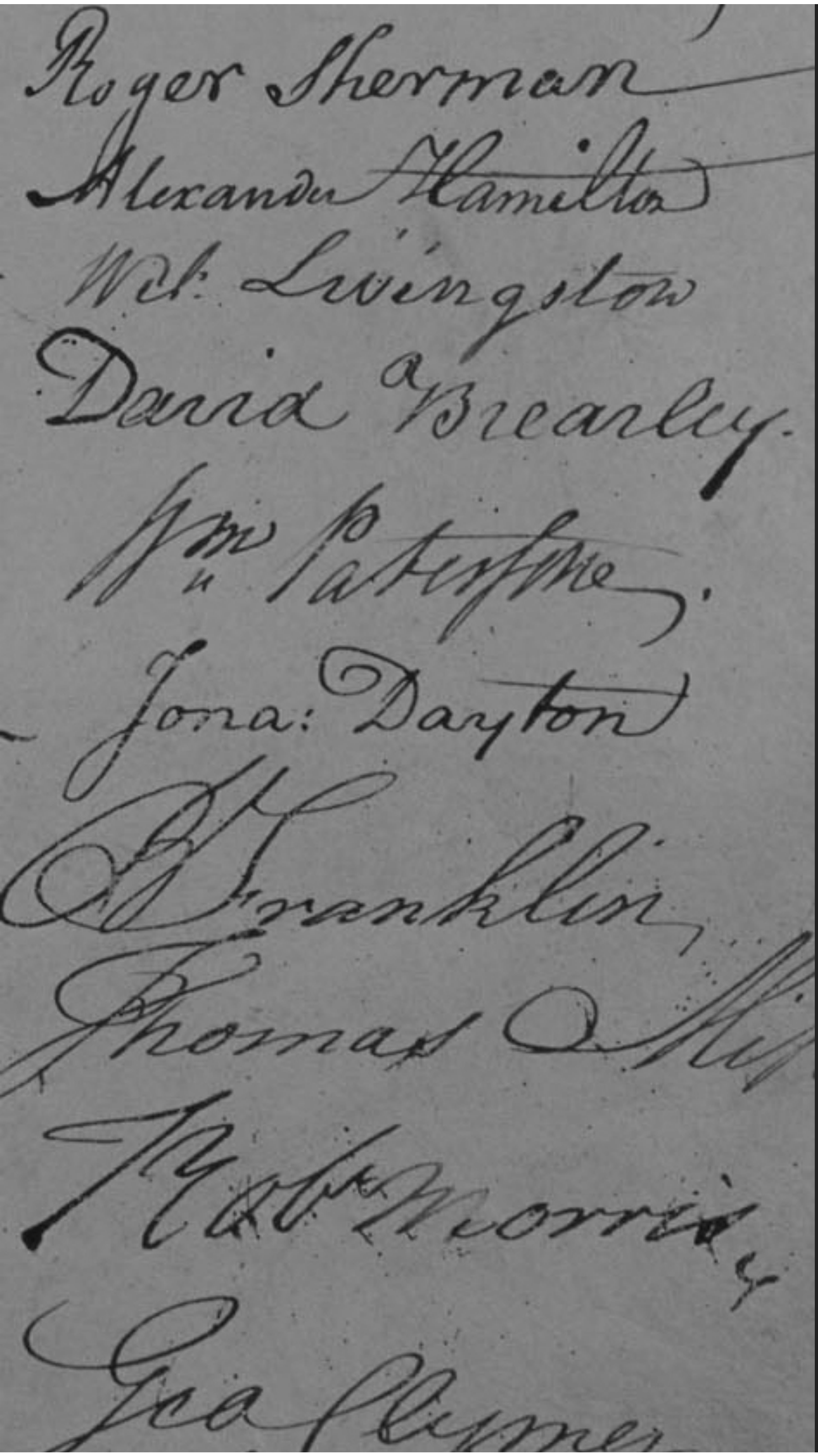
- Replacing an old transaction with a new version with a higher transaction fee
- Solves problem of transaction becoming ‘stuck’ due to low fee
- Miner can choose to include any version of a transaction
- However, most nodes won’t allow transactions to be replaced in general

OPT-IN REPLACE-BY-FEE

- ‘Opt-in’ RBF is defined in BIP 125
- Allows users to signal that their transaction can be replaced later
- Uses the the sequence number in one of the transaction inputs
- If the sequence number is < 0xfffffffe, the transaction is ‘replaceable’

OPT-IN REPLACE-BY-FEE CONDITIONS

- RBF could potentially be used as a DoS vector
- RBF replacement transaction will be accepted if:
  1. One input from original transaction has sequence < 0xfffffffe
  2. The replacement transaction pays a higher fee than the sum paid by the original transactions
  3. The replacement transaction does not contain any new unconfirmed inputs
  4. The replacement transaction ‘pays for its own bandwidth’
  5. The replacement transaction does not replace more than 100 transactions



SIGNATURE AND SCRIPT CACHING

TRANSACTION CACHING

- Most transactions are seen twice:
  1. When they are propagated and accepted to the mempool
  2. When they are included in a block
- Rather than fully validate the transaction twice, we can partially cache the result of the first validation
- Caching results makes block validation *much* faster

SIGNATURE CACHING

- Signature validation is the most expensive part of transaction validation
- Usually, each transaction input involves at least one ECDSA signature validation
- Keep a cache of signature evaluations
- Added to Bitcoin Core in v0.7.0

SCRIPT CACHING

- Bitcoin script validity is *context-free*. It doesn’t depend on any data outside the transaction
- As well as caching the validity of the signature, we can cache the validity of the entire scriptSig in each transaction input
- Added to Bitcoin Core in v0.15.0

TRANSACTION CACHING?

- Why not cache the validity of the entire transaction?
- Transactions **are** contextual. Validity depends on data outside the transaction

MEMPOOL AND SCRIPT CACHING

- Signature/Script caching requires a mempool
- Node needs to be online for some period of time
- a **-blocksonly** node won’t benefit from signature/script caching
- Signature/Script caching can be though of as front-loading block validation

COMPACT BLOCKS

COMPACT BLOCKS

- Saves block propagation bandwidth and time
- Doesn’t include all the raw transactions in the block
- Defined in BIP 152
- Two versions:
    - low bandwidth - saves on block propagation bandwidth
    - high bandwidth - saves on block propagation time

COMPACT BLOCKS AND THE MEMPOOL

- New P2P message **CMPCTBLOCK**
- Does not contain all full transactions
- Excludes transactions that the transmitting node thinks the receiving node has already seen
- Refers to transaction by shortid (6 bytes digest using SipHash-2-4)
- Receiving node reconstructs block using transactions in its mempool
- Receiving node can request missing transactions with **GETBLOCKTXN**

FEE ESTIMATION

FEE ESTIMATION

- Bitcoin transactions include an implicit transaction fee
- Miners chose which transactions to include based on fee rate
- Prevailing fee rate depends on current level of demand for blockchain space
- Estimating how much fee is required is a hard problem

USING THE MEMPOOL FOR FEE ESTIMATIONS

- Looking at a snapshot of the mempool gives an idea of the current ‘competition’
- However, there are problems with just using the mempool:
  - Expected time to wait for a block is always 10 minutes
  - Doesn’t account for ‘lucky’/‘unlucky’ runs
  - There’s no such thing as *the* mempool

USING RECENT BLOCKS FOR FEE ESTIMATION

- Looking at recent blocks gives an idea of what fees were required for a transaction to be included in a block
- However, this is trivially gameable by miners
- A miner could stuff his block with private, high fee paying transactions to break users’ fee estimates

USING RECENT BLOCKS AND THE MEMPOOL

- Bitcoin Core therefore considers the historic data for transactions in the mempool *and* in recent blocks
- Requires a large enough sample of recent transactions to make a good estimate
- Requires node to be running for some time with a mempool