An in-depth guide into how the mempool works

Marion Deneuville Follow
Oct 12, 2016 · 4 min read

The mempool stores transactions waiting to be validated by miners

To submit a transaction to the miners, nodes have to relay it to each other until it has propagated across the entire network. **The mempool is the node's holding area for all the pending transactions.** It is the node's collection of all the unconfirmed transactions it has already seen enabling it to decide whether or not to relay a new transaction.

There are as many mempools as there are as nodes

As the Bitcoin network is distributed, not all nodes receive the same transactions at the same time so some nodes store more transactions than others at some time. Plus, everyone can run its own node with the hardware of his choice; so all nodes have a different RAM capacity to store unconfirmed transactions. As a result, **each node has its own rendition of the pending transactions,** this explains the variety of Mempool sizes & transactions counts found on different sources.

How do transactions get into the Mempool?

Before letting a transaction into its Mempool, a node has to complete the series of checks listed below.

```
Check syntactic correctness

Make sure neither in or out lists are empty
```

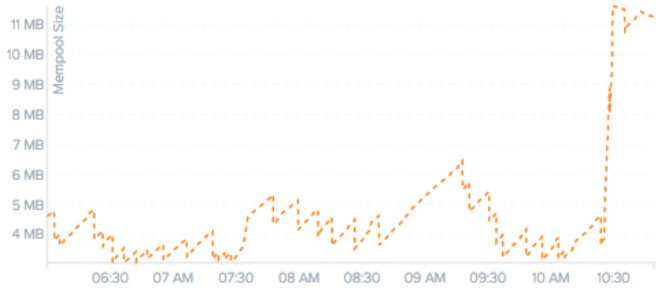
```
Size in bytes < MAX_BLOCK_SIZE

Each output value, as well as the total, must be in legal
money range

Make sure none of the inputs have hash=0, n=-1 (coinbase
transactions)

Check that nLockTime <= INT_MAX, size in bytes >= 100, and
sig opcount <= 2

Reject "nonstandard" transactions: scriptSig doing anything
other than pushing numbers on the stack, or scriptPubkey not
matching the two usual forms

Reject if we already have matching tx in the pool, or in a
block in the main branch

For each input, if the referenced output exists in any other
tx in the pool, reject this transaction.

For each input, look in the main branch and the transaction
pool to find the referenced output transaction. If the
output transaction is missing for any input, this will be an
orphan transaction. Add to the orphan transactions, if a
matching transaction is not in there already.

For each input, if the referenced output transaction is
coinbase (i.e. only 1 input, with hash=0, n=-1), it must
have at least COINBASE_MATURITY (100) confirmations; else
reject this transaction

For each input, if the referenced output does not exist
(e.g. never existed or has already been spent), reject this
transaction

Using the referenced output transactions to get input
values, check that each input value, as well as the sum, are
in legal money range

Reject if the sum of input values < sum of output values

Reject if transaction fee (defined as sum of input values
minus sum of output values) would be too low to get into an
empty block

Verify the scriptPubKey accepts for each input; reject if
any are bad

Add to transaction pool
```

```
"Add to wallet if mine"

Relay transaction to peers

For each orphan transaction that uses this one as one of its
inputs, run all these steps (including this one) recursively
on that orphan
```

If the transaction matches the above criteria, it is allowed into the Mempool and the node starts broadcasting it. If it doesn't the transaction is not re-broadcast by the node.

How does a new block impact the Mempool?

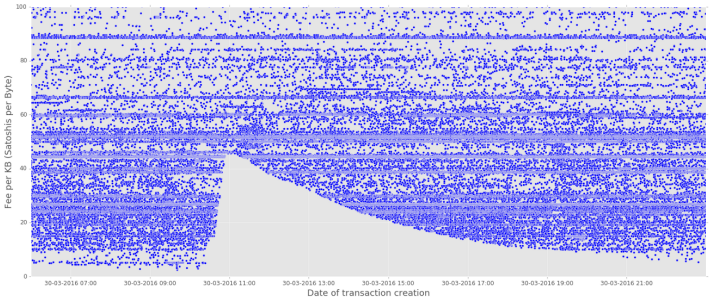
When a node receives a new valid block, it removes all the transactions contained in this block from its mempool as well as the transactions that have conflicting inputs. This results in a sharp drop in the Mempool size:



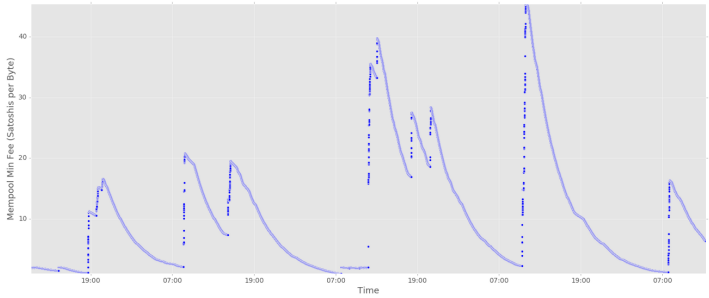
What happens when the node's memory get full?

Unlike mining, there is no financial incentive for running a node. Therefore, the hardware dedicated to it tends to be limited and so a node's Mempool often max out its RAM. When this happen, in former versions of bitcoind, the node would just crash and restart with an empty Mempool.

In recent versions of bitcoind (0.12+), if the Mempool size gets too close to the RAM capacity, the node sets up a minimal fee threshold. Transactions with fees per kB lower than this threshold are immediately removed from the Mempool and only new transactions with a fee per kB large enough are not allowed access to the Mempool.



Over time the node decreases its fee threshold to put it back to the node's **minrelayfee**. This threshold can be monitored using the **getmempoolinfo** RPC command. Here is a snapshot of the mempool min fee over a random period of time:



Mempool stats are available via our API on Mashape.