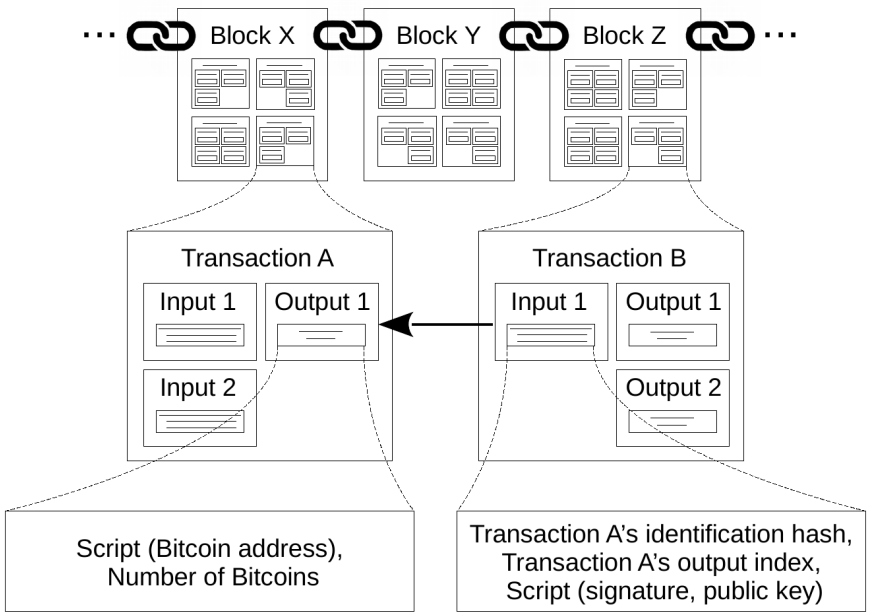
Cuckoo's Nest: An Ultra-Lightweight DoS-Resilient Bitcoin Mempool

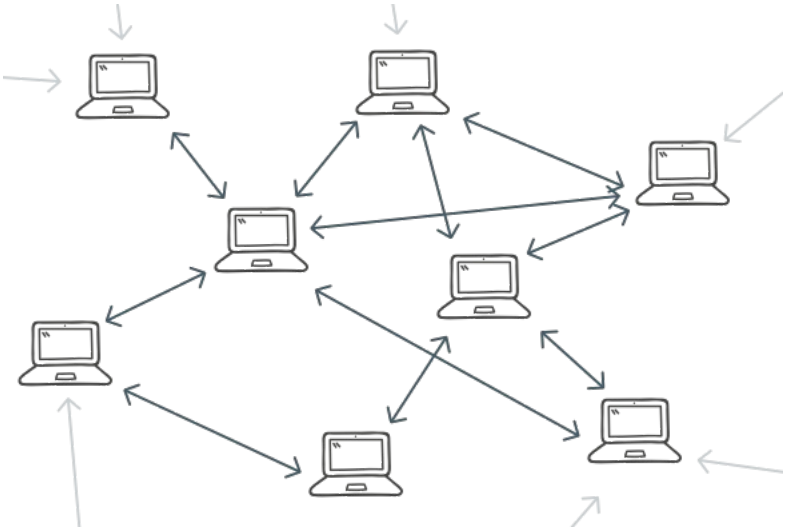
Hina Binte Haq, Syed Taha Ali, Siamak F. Shahandashti

Paper ePrint

The Context

Overview of Bitcoin





- Transactions (Tx's): identified by TxHash
- Spendable inputs: identified by inputTxHash & index
- Unspent Tx Outputs (UTXO): computed from Blockchain

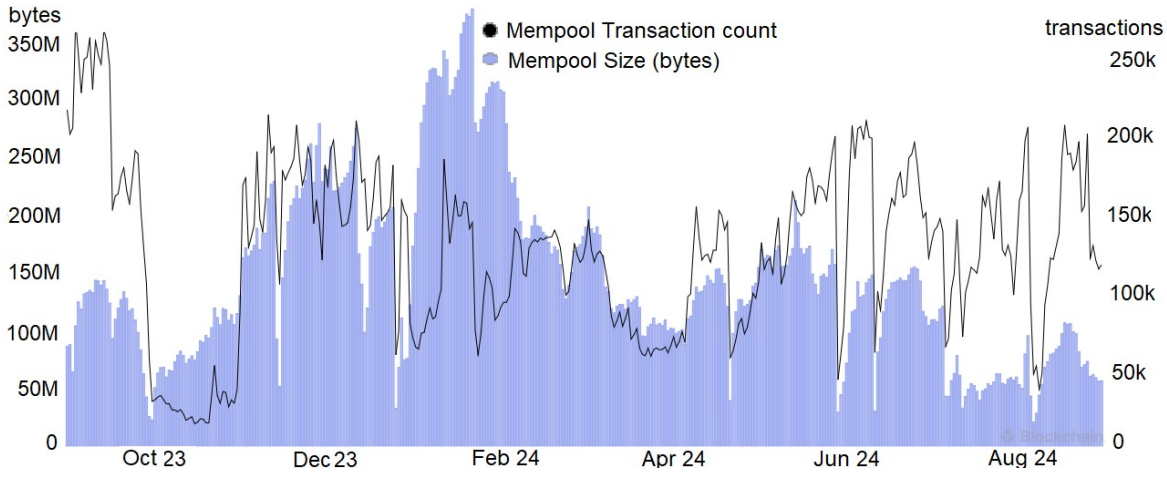
- P2P network gossips Tx's to achieve consensus
- Clients inc. miners & other lightweight clients, e.g. SPV
- 2 functions: inventory (miners), forwarding (all)

Bitcoin Client Memory Pool (Mempool)

- Receive TxHash, Lookup, Request Tx
  - Store TxHashes: mempool’s mapTx in Bitcoin Core
- Verify Tx, Check for double spend against UTXO & circulating Tx’s
  - Store Tx inputs: mempool’s mapNextTx in Bitcoin Core
- Remove Tx if included in a block, Tx expiry, reaching capacity
  - Default mempool capacity: 300 MB

Mempool Size

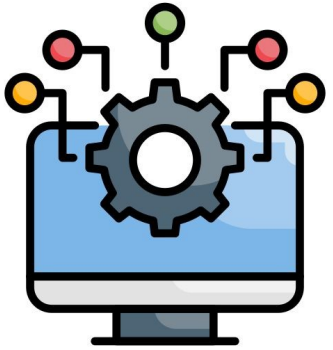
- Bitcoin nodes regularly work at near capacity
- Vulnerable to “dust” attacks: flood of small-value transactions
- e.g. in 2015, Bitcoin mempool 1 GB, 10% nodes crashed



Problem

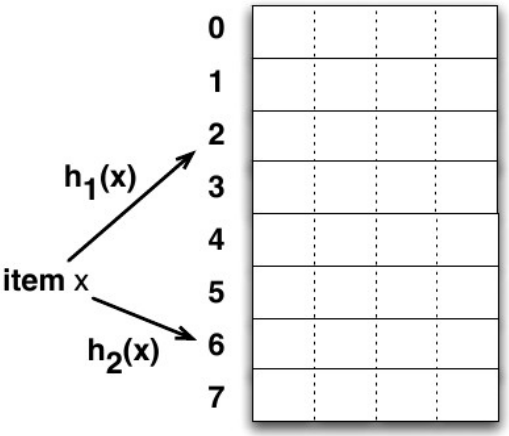
- Client memory consumption needs to be reduced
  - Pruned, NIPoPoW, FlyClient, TxChain reduce bootstrapping
  - Dietcoin, Utreexo compress the UTXO
  - Contra, Anti-Dust aim to identify, remove dust transactions
  - Not much work on mempool
    - Probabilistic data structure for membership testing
    - Our previous work: Neonpool, Carbyne using Bloom filters
    - This work: Cuckoo’s Nest using Cuckoo filters

The Design



Cuckoo Filter

- Probabilistic data structure for membership testing
- Similar underlying idea to Bloom filters
  - Allows deletion
- m buckets × b slots of f bits
- Uses less space than Bloom filters, supports deletion, when FPR < 3%



Bitcoin Mempool with Cuckoo’s Nest

- Store TxHashes: mapTx in Bitcoin Core
  - Cuckoo’s Nest: CuckooTxFilter
- Store Tx inputs: mapNextTx in Bitcoin Core
  - Cuckoo’s Nest: CuckooTxInputsFilter
- Remove Tx if
  - included in a block: remove from CuckooTxFilter
  - Tx expiry: ?
  - reaching capacity: ?

Empirical Testing Conditions

- Instrumented client implemented in C++
- Dataset: 30 million unique Tx’s from 90 million Tx announcements over 30 days
- Replay dataset to Bitcoin Core and Cuckoo’s Nest

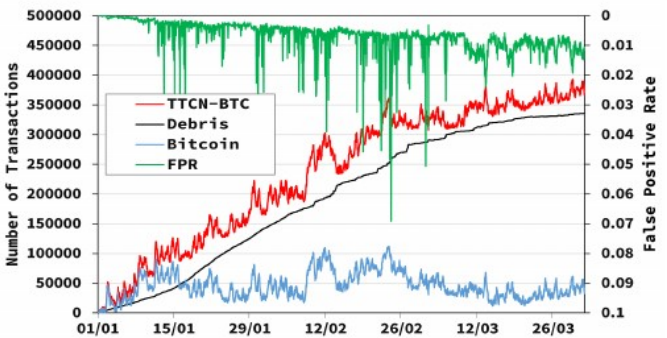
Empirical Testing Metrics

- Three events: inventory (in), entry (en), exit (ex)
- False positives are generally undesirable
  - Overall FPR = $(FP_{in}+FP_{en}+FP_{ex}) / (Q_{in}+Q_{en}+Q_{ex})$
- False positives at in or en result in discarded transactions
  - Discard Rate = $(FP_{in}+FP_{en}) / (Q_{in}+Q_{en})$
- Inventory false negatives result in reprocessed transactions
  - Reprocessing Rate = FN_{in} / Q_{in}

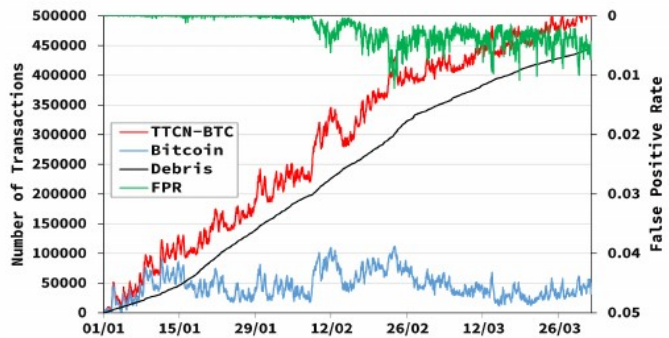
The Results

Using A Single Cuckoo Filter

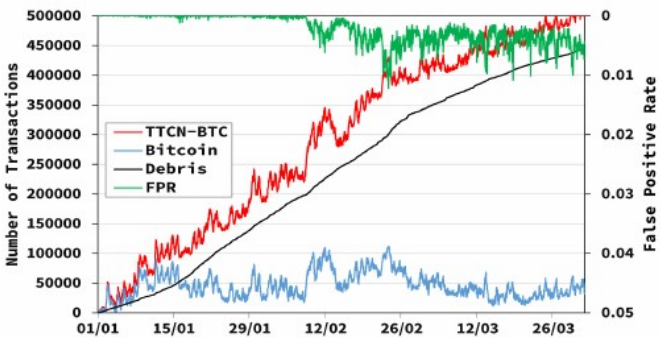
- False negatives mean Tx's erroneously added to or not removed from mempool, leading to false positives
- We call this debris



(a) CuckooTxFilter= 1 MB



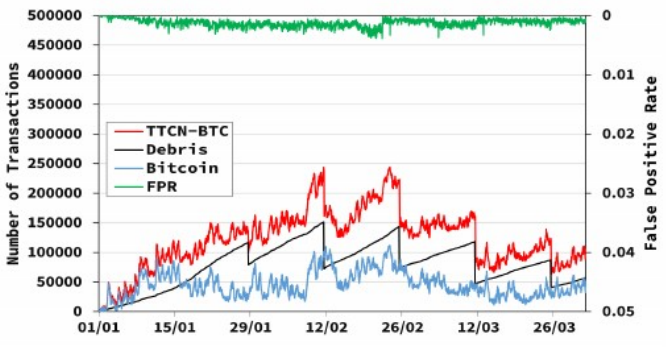
(b) CuckooTxFilter= 2 MB



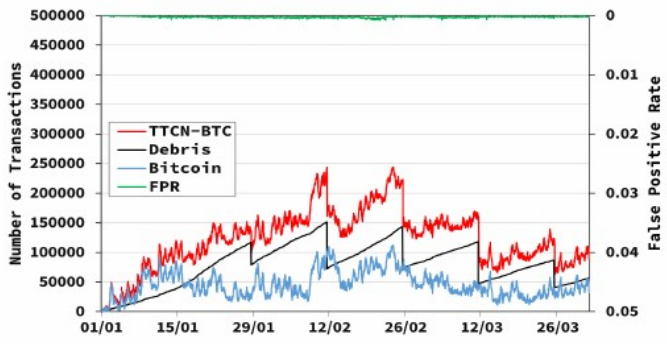
(c) CuckooTxFilter= 4 MB

Using 2 Cascade Filters

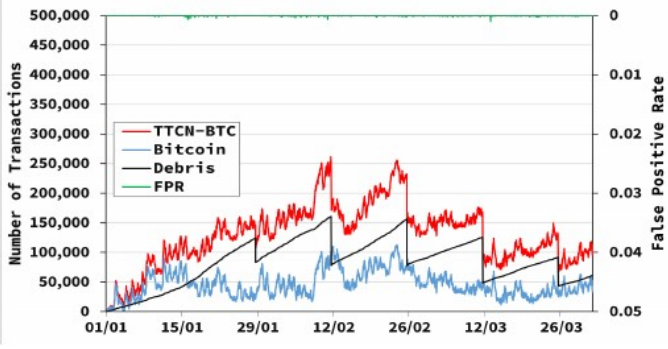
- Let’s use 2 filters, periodically clearing one and then the other
  - Using Bitcoin’s default 14-day expiry period
- Debris, hence false positives kept down



(d) CuckooTxFilter= 1 MB×2



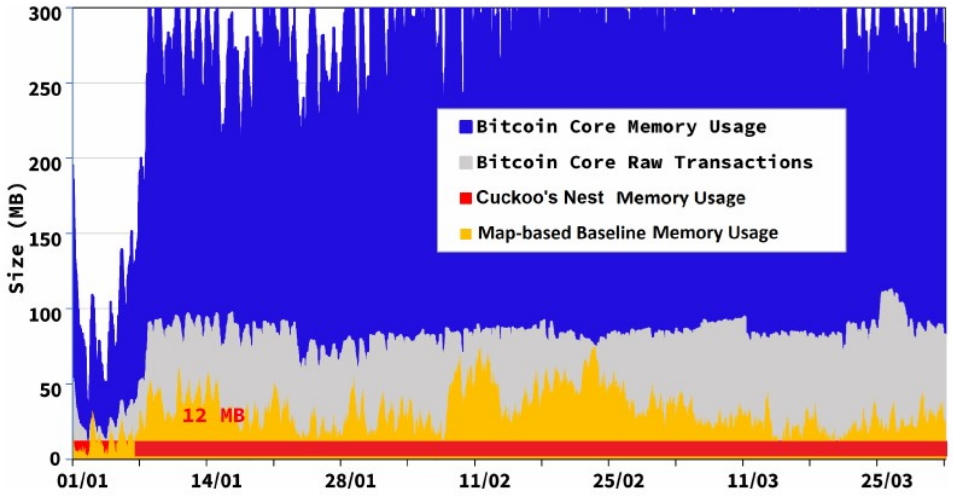
(e) CuckooTxFilter= 2 MB×2



(f) CuckooTxFilter= 4 MB×2

Memory Footprint

- Goal: discarding only 0.001% of Tx's, i.e. achieving 99.999% accuracy
- Requires 3×4 MB filters (2×Tx+1×TxInputs filters), i.e. 12 MB memory



Resilience against Dust Attacks

- Hash functions use a long (e.g. 128-bit) salt
- Injecting or censuring transactions at individual nodes is improbable
- Network-wide injection or censuring is even harder as salts are independent
- False negatives only cause reprocessing, not double-spends

Conclusions

- Cuckoo’s Nest reduces node memory consumption at the expense of little loss in accuracy of forwarding
- Straightforward to adapt to Bitcoin derivatives like Bitcoin Cash, Bitcoin Gold, Litecoin, Dogecoin
- Future challenges
  - Adapting for account-based systems like Ethereum
  - Bootstrapping new nodes

Thank you.

Siamak F. Shahandashti
cs.york.ac.uk/~siamak

@siasha.bsky.social