# Performance Impacts of Raising the Ancestor and Descendant Transaction Limit for Bitcoin Cash

David Foderick (dfoderick@gmail.com, @FullCycleMining Telegram)
Feedback welcome in Bitcoin Testers telegram group
https://t.me/joinchat/Hu1_WxJLvSBIseBEt7gCkA

## Purpose

Many popular Bitcoin Cash applications are negatively impacted by the current default limitation of 25 transaction descendants and ancestors [1]. Most participants in the community believe the current setting of 25 descendants and ancestors to be unnecessarily restrictive, as it was added by the Core developers long ago to prevent network spam [2]. Bitcoin Cash protocol developers agree that this limit could probably be raised, but to what extent is unknown. The purpose of this study is to gather the data necessary to enable full node operators to make an educated decision for how high the descendant and ancestor limit could be raised without risking network security.

The descendant and ancestor limits are configurable settings which can be adjusted using command line arguments or the bitcoin.conf file. The settings to be tested are:

- `limitancestorcount=<n>`
- `limitancestorsize=<n>`
- `limitdescendantcount=<n>`
- `limitdescendantsize=<n>`

## Testing Setup

1. Bitcoin ABC was forked for the purpose of adding proper instrumentation/telemetry and python tests to the repo https://github.com/jcramer/bitcoin-abc

2. A virtual machine was configured for easily sharing the test environment with the team. The process for provisioning the test environment is documented here. https://docs.google.com/document/d/182clkcsNOlTD7JgyfxwjsBWgls1Y5kCBBP1F1nVyvYY/edit#
   The virtual machine was configured for 2 CPU and 8 GB of RAM.

**bitcoin.config settings**

regtest=1

## Testing Cases

The following settings were used to accommodate all test conditions.

```
-limitancestorcount=2000
-limitdescendantcount=2000
-limitancestorsize=1000
-limitdescendantsize=1000
```

Mempool acceptance was tested using a synchronous call to send transactions to the node one by one.
`txid = node.sendrawtransaction(signedtx['hex'])`

Validation time was tested using GetBlockTemplate which assembles a block and validates the transactions within.
`templat = self.nodes[0].getblocktemplate()`

The test harness can be found in
https://github.com/jcramer/bitcoin-abc/blob/master/test/functional/chained_transactions.py

### Running the tests

An individual test can be run using the chained_transaction.py script where the test parameter is the number of chained transactions in a range from 10 to 2000 (50 in the example below).

```
export SRCDIR=~/bitcoin-abc
cd ~/bitcoin-abc/test/functional
python3 chained_transactions.py 50
```

This will produce detailed node debug output using the -printtoconsole parameter.

A formatted test suite can be run using the ./run_chained.sh shell script in the same directory.

## Testing Results

The following table contains the actual results observed by adding between 10 and 2000 chained transactions to the mempool and then validating that chain of transactions.

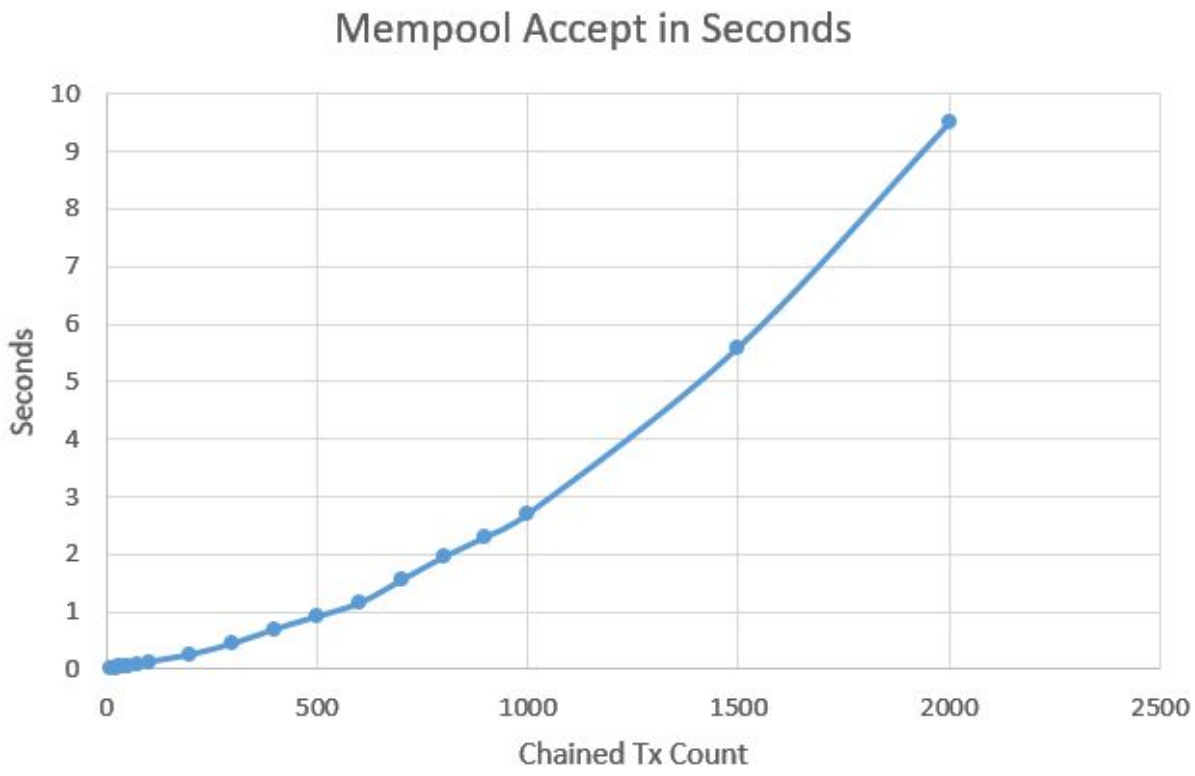
Please note that the following results represent the worst possible case for chained transactions where all the transactions are in one long chain. Certainly this is not a normal block with a mix of chained and unchained transactions.

| **Chained Tx Count** | **GetBlockTemplate (Sec)** | **Mempool Accept (Sec)** | **Mempool Size (Bytes)** |
|---|---|---|---|
| 10 | 0.0004 | 0.01128 | 4244 |
| 25 (Current Limit) | 0.0009 | 0.03132 | 10721 |
| 30 | 0.0011 | 0.038 | 12880 |
| 35 | 0.0013 | 0.04402 | 15043 |
| 45 | 0.0018 | 0.052 | 19353 |
| 50 | 0.0021 | 0.057 | 21528 |
| 75 | 0.0043 | 0.09 | 32333 |
| 100 | 0.0076 | 0.148 | 43132 |
| 200 | 0.0249 | 0.29 | 86322 |
| 300 | 0.0740 | 0.45 | 129524 |
| 400 | 0.1275 | 0.7 | 172708 |
| 500 | 0.2141 | 0.95 | 215902 |
| 600 | 0.3236 | 1.2 | 259100 |
| 700 | 0.4580 | 1.5 | 302296 |
| 800 | 0.6288 | 1.95 | 345516 |

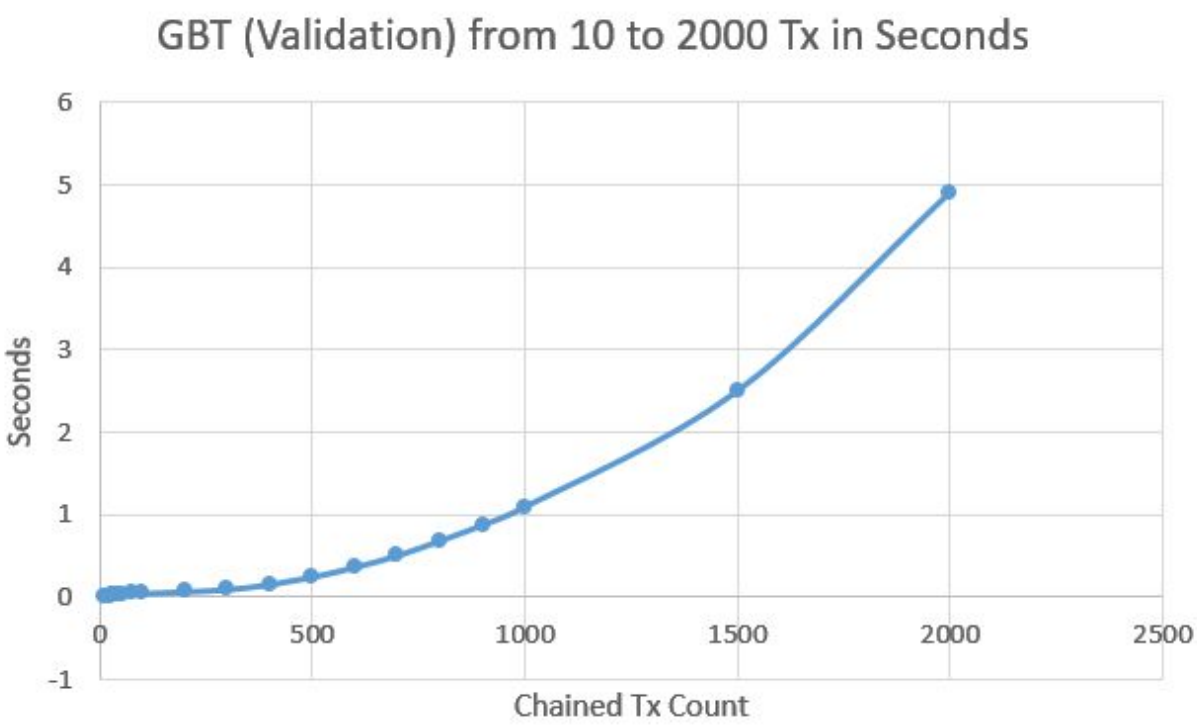
| | | | |
|---|---|---|---|
| 900 | 0.8209 | 2.3 | 388718 |
| 1000 | 1.0062 | 2.8 | 431916 |
| 1500 | 2.3556 | 5.7 | 647928 |
| 2000 | 3.9026 | 9.5 | 863968 |

Please review the following graphs with the understanding that these are intended to show the relative performance of varying the number of transactions in the transaction chain. Certainly better performance could be observed by running the tests on more powerful hardware but that is not the intent of this study.

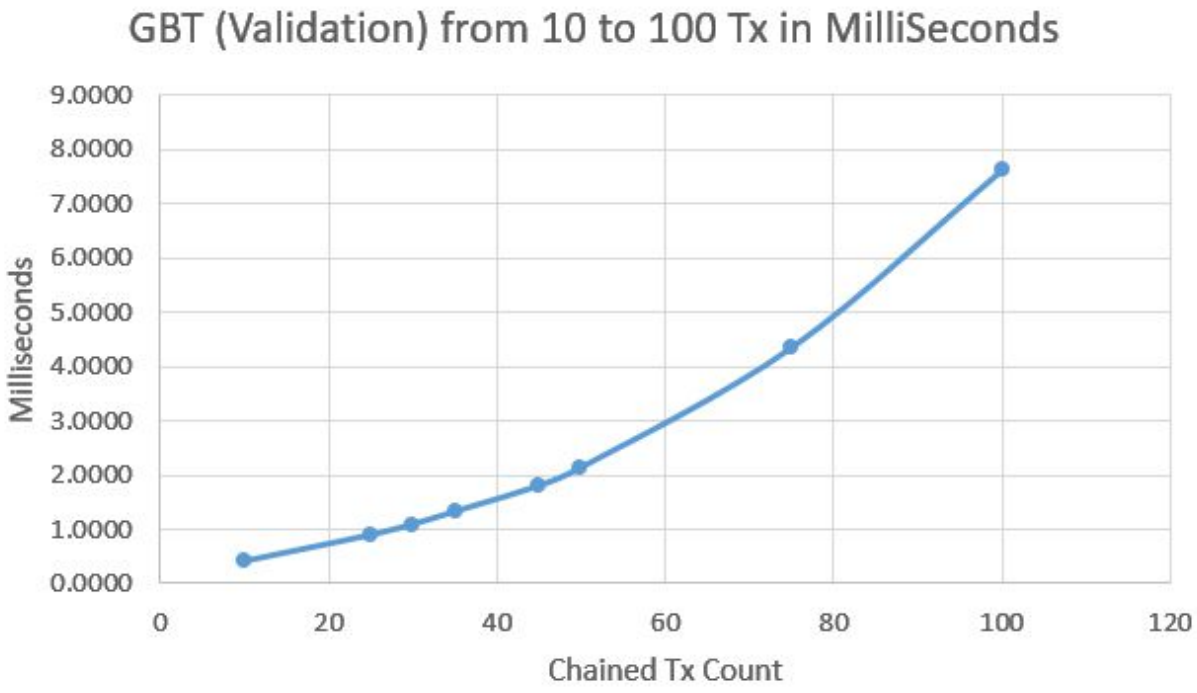
Mempool Acceptance shows the time that it took to load the local node's mempool with the chained transactions. The mempool tracks and manages chained transactions.




GetBlockTemplate is the method call that takes transactions from the mempool and assembles them into blocks. The following graph shows the results over the complete data set (10 to 2000 transactions).




This graph shows an exploded view of the same data but only from 10 to 100 transactions.



### Why are these performance graphs Exponential instead of Linear?

The performance bottleneck for chained transactions is in the management of the mempool. Transaction dependencies have to be tracked as well as fees for the whole chain in the case of Child Pays For Parent (CPFP).

The resulting complexity of the data structure in the mempool leads to multiple breadth-first search (BFS) executions. In the near future, the code that manages transactions in the mempool can be optimized to allow for even larger ancestor and descendant limits without incurring a large performance penalty.

# Conclusion

Full node operators should prepare for a new class of applications that produce a long chain of transactions. By reviewing these graphs they can decide what point on the curve would be acceptable for performance.

Node operators can accept these chained transactions by increasing the following bitcoind parameters:

```
-limitancestorcount=<n>
-limitancestorsize=<n>
-limitdescendantcount=<n>
-limitdescendantsize=<n>
```

Even a modest increase in the transaction limit would be a huge gain for application developers on the BCH chain and would assist in the quick adoption of Bitcoin Cash.

**References**

[1] - Chained 0-Conf Transactions on Memo.
https://jasonc.me/blog/chained-0-conf-transactions-memo

[2] - Bitcoin Core 0.12.0 Release Notes - Memory Pool Limiting.
https://github.com/bitcoin/bitcoin/blob/57b34599b2deb179ff1bd97ffeab91ec9f904d85/doc/release-notes/release-notes-0.12.0.md#memory-pool-limiting