vExtraTxnForCompact Considered Useful

tl;dr

The `bitcoin.conf` option `blockreconstructionextratxn` is underappreciated, especially in the context of heterogeneous node policies, and more noderunners should consider increasing it from its current default. On a node that applies restrictive spam-filtering policies, a substantial number of filtered transactions can be recovered from `vExtraTxnForCompact`, saving the need to request them from peers during compact block reconstruction.

Motivation

An underappreciated feature of Bitcoin Core is `vExtraTxnForCompact`, which is a sort of purgatory where a hodgepodge of misfit transactions swirl around a ring buffer before being shuffled off into the void. Here, transactions replaced in the mempool, orphans, and transactions rejected for policy reasons await a last chance at redemption during compact block reconstruction. If they get mined, the node uses them to reconstruct the block instead of requesting them from its peers.

Poor little vExtra doesn't get much respect, having been implemented opportunistically in a way that isn't especially robust. And, by default, only 100 transactions are buffered. Consequently, it seems, a lot of folks engaged in the recent brouhaha around `OP_RETURN` forgot about (or were never aware of) its very existence. If nodes filter moby datacarriers (and other spam), the thinking went, they won't have those transactions around to reconstruct blocks and will have to request them from their peers. But, in fact, those transactions may be present in vExtra.

Even for nodes running with Core's defaults (i.e., `datacarriersize=83`) and a modest increase of `blockreconstructionextratxn` from its default of `100` to `1000`, as 0xB10C reported last year, vExtra improves compact block reconstruction performance. But vExtra's potential is even greater for nodes that apply more restrictive policies. For nodes running Bitcoin Core, that might mean `datacarriersize=0` (for now, at least) and `permitbaremultisig=0`. For nodes running Bitcoin Knots, which uses the same code to handle vExtra but allows more control over policy, there's potentially a much larger number of rejected transactions added to vExtra.

Sjors recently proposed a (temporary?) change that would only put replaced transactions, not transactions rejected by policy, in vExtra. Despite his valid concern that vExtra isn't robust to attacks that could fill it with garbage, consuming memory while also making it useless, this seemed to me like a big step backward at a time when many are concerned about degredation of compact block reconstruction. But exactly how load-bearing has this modest little data structure become? I decided to try a simple experiment to get a better sense of how helpful vExtra can be in the face of heterogeneous node policies.

Method

My strategy was to run a filtering node and a non-filtering node side-by-side and compare how many transactions they requested during compact block reconstruction and how many they retrieved from vExtra, as follows:

- Both nodes running Knots (Satoshi:27.1.0/Knots:20240801) on identical hardware on a 1 gbps connection (with IPv4, IPv6, and Tor) in the same subnet.
- Both nodes with `maxmempool=1200`, starting at the same time (shortly before block 897568) with an empty mempool.
- Both nodes with `blockreconstructionextratxn=100000`. (This seemed like the largest value that would keep memory usage from growing too large in the presence of an attack, and it seemed unlikely that larger values would greatly increase performance.)
- Both nodes connected to a node with Core 29.0 and default settings plus one running Core 29.0 with `datacarriersize=400000` for good measure.
- The two nodes peered with each other using `whitelist=forcerelay,in,out`.
- The equivalent of current Core defaults (including `datacarriersize=83`) on the non-filtering node.
- The following aggressive policy on the filtering node:

```
acceptnonstddatacarrier=0
acceptnonstdtxn=0
datacarrier=1
datacarriercost=1
datacarriersize=83
permitbaremultisig=0
permitbarepubkey=0
rejectparasites=1
rejecttokens=1
```

I collected 145 blocks of data (blocks 897568 through 897712), during a period when the network was reasonably active (only one empty block, not due to lack of transactions) and there were a fair number of inscriptions (mostly small BRC-20 transactions) being mined.

Results

In total, the filtering node requested 4% of the transactions needed for compact block reconstruction, vs. 1% for the non-filtering node. So, as expected, the ~101,000 additional transactions the filtering node rejected due to its more-restrictive policies resulted in almost triple (16,598 vs. 6,115) the requests from peers.

However, performance on the filtering node would have been far worse without vExtra! Of the 415,686 transactions in these blocks, the filtering node found

103,820 (25%) in vExtra vs. only 24,413 (6%) for the non-filtering node. Put another way, the filtering node recovered about 80% of the filtered transactions from vExtra at reconstruction time.

Conclusions

Despite valid concerns about its robustness, `vExtraTxnForCompact` is earning its keep in Bitcoin Core (and, by extension, Knots). Any node that isn't severely resource-constrained should consider increasing `blockreconstructionextratxn` from its default of `100`, and miners—especially those filtering spam—should probably consider a much higher value. (Only rejected transactions <100kb are added to vExtra, so, despite being sized by the number of entries rather than by the amount of memory, its size is still limited.)

Someone who is more skilled at C++ than I (or who has more time to trace individual transactions from blocks back to log file entries) might consider collecting more data about which types of rejected transactions (and other denizens of vExtra) eventually find their way into blocks, and should perhaps be prioritized. Even simply providing options to configure what does and does not get added to vExtra might be useful to noderunners with different needs.

Given its potentially increasing importance, making vExtra less susceptible to attack and/or engineering a new approach that appropriately handles the different types of transactions currently added to vExtra seems like a worthy goal that would improve node performance.

– szarka (2025-05-21)