Project Report

Cyrus Malekpour (cm7bv)

Motivation

The motivation behind my project was the large amount of disk space needed for full nodes. Since the course started, the blockchain has increased in size from roughly 40gb to 48gb. While this is a relatively minor amount of disk space given current prices per gigabyte, it makes operating a full node or multiple full nodes difficult for small, portable, or most embedded devices. It also means that the download during the initial sync of the blockchain is extremely long and makes starting up new full nodes time intensive. Currently, it can take multiple days on an average network connection to sync to the head of the blockchain. If Bitcoin is to stay decentralized and secure, it should be easier for new full node operators to enter the network.

Given that full nodes are vital to the health of the bitcoin network, I was surprised to find how difficult in practice it can be to operate one. Over the past 90 days, the number of reachable full nodes on the network has dropped ~19% to around 5529 [1]. Looking to the future, the recent block-size debate may lead to further increases in the size of the blockchain, and more further pressure on full node operators.

Background

Once a Bitcoin full node has created its set of unspent transactions outputs (UTXOs), it is able to validate new blocks without accessing older block data in the blockchain. Only when the network forks and the client chooses a chain that becomes the minority must it look up older blocks to revert its state. If we can loosen the requirement that full nodes store the complete history of past transactions, we would only need the UTXO set and a few buffer blocks to operate the node. In other words, we no longer need to store the whole blockchain.

For this project, I define pruning nodes as nodes that only store the latest *n* blocks in the blockchain, where n is a small number relative to the total height of the chain. When there are at least *n* blocks stored locally, blocks are deleted oldest-first as new blocks come in. The blocks themselves are stored only to provide a buffer in the case of a network fork. One common recommendation is to wait 6 confirmations for safety. However, with a current max blocksize of 1 megabyte, we can store 100 or 500 blocks even on space restricted devices.

Currently, Bitcoin Core relies on a series of hardcoded block hashes and heights as "checkpoints" for its blockchain download. These are used to limit how much of the transaction history a theoretical 51% attacker could rewrite. Though these checkpoints have been somewhat controversial in the community, we can use the latest checkpoint hash as a new "genesis block" and only download blocks after it in the chain for a much faster sync. Even as we download we can prune the number of saved incoming blocks so we never exceed *n* on disk.

Pruning is currently implemented as an option in Bitcoin Core (the `--prune` flag) [2]. This implementation operates as specified above, with several restrictions. A minimum of 550mb of block data must be stored (~288 blocks on disk). Nodes with `--prune` enabled also will not relay blocks to other nodes, or provide wallet functionality to their operators. Some of these limitations, such as disabled wallet functionality, are due to the fact that old blocks can't be scanned to find wallet history. However, disabled block relay and block storage minimums are not requirements of the pruning method. Satoshi's whitepaper also mentions another method of pruning by discarding old transactions [3]. Satoshi describes discarding transactions in old blocks by removing transactions and hashes from the merkle tree. However, this method still involves storing 80 bytes per block, rather than a fixed disk size for the whole chain.

Another common "light client" implementation is Simple Payment Verification (SPV), which works by checking if the transaction's block has a sufficient number of confirmations. SPV wallets store only the block headers for each block, and use the # of confirmations provide full wallet functionality to users [4]. These clients are common on smartphones or other disk/network restricted devices. SPV clients were also initially described in Satoshi's whitepaper [5]. While SPV clients use significantly less disk/network space than both full nodes and pruning nodes, their disk space still increases linearly as the blockchain grows. Since they can't verify blocks for themselves, they must wait for more confirmations (30+ recommended) and can be affected by sybil attacks, where all the nodes they connect to are attacker controlled. Furthermore, there are concerns about the scalability of current implementations of bloom filters for SPV clients [6].

What I did

For my project, I modified the btcd full node software to operate as a pruning node. btcd has the benefit of being only a full node (no wallet software), which means the code was simpler to integrate. My version of the software now uses the latest checkpoint hash/height to request blocks from fellow nodes, and avoids the initial transaction checking as it syncs with the blockchain. Instead, it only verifies block headers during the initial sync, similar to SPV, and relies on other nodes to do the more intensive verification work for it.

Once it has downloaded *n* blocks (*n* = 50 in the configuration for testing purposes), it begins pruning old blocks. btcd already constructs the UTXO set separately from the blocks, so we can delete old blocks without causing any problems.

Results

My client now operates successfully as a pruning node. The pruning code did not cause any loss in verification functionality as a full node, although the client is now unable to help other nodes retrieve older blocks. It also causes issues with SPV, since we don't store some blocks SPV clients are concerned with looking up.

The initial sync code also works and downloads only the blocks that occur after the latest checkpoint. However, since we initially can't verify the transactions in those blocks (we haven't seen the addresses before), we have to assume they are valid. To address this in the future, we would trace more transaction outputs to the set of blocks we have seen and thus be able to verify them. This compromise is similar to that which SPV makes, but because we would try to verify whenever we can, over time we would approach a complete/correct UTXO set.

Future work would also involve tracking down some latent code that attempts to check for older blocks, further testing the impact of my changes on SPV, and investigating what btcd actually stores on disk to see if further improvements are possible.

References

[1] https://bitnodes.21.co/dashboard/?days=90

[2] https://bitcoin.org/en/release/v0.11.0

[3] https://bitcoin.org/bitcoin.pdf (section 7)

[4] https://en.bitcoin.it/wiki/Thin*Client*Security#Simplified*Payment*Verification.*28SPV.29*Clients

[5] https://bitcoin.org/bitcoin.pdf (section 8)

[6] https://bitcoin.stackexchange.com/questions/37344/is-the-spv-client-model-scalable