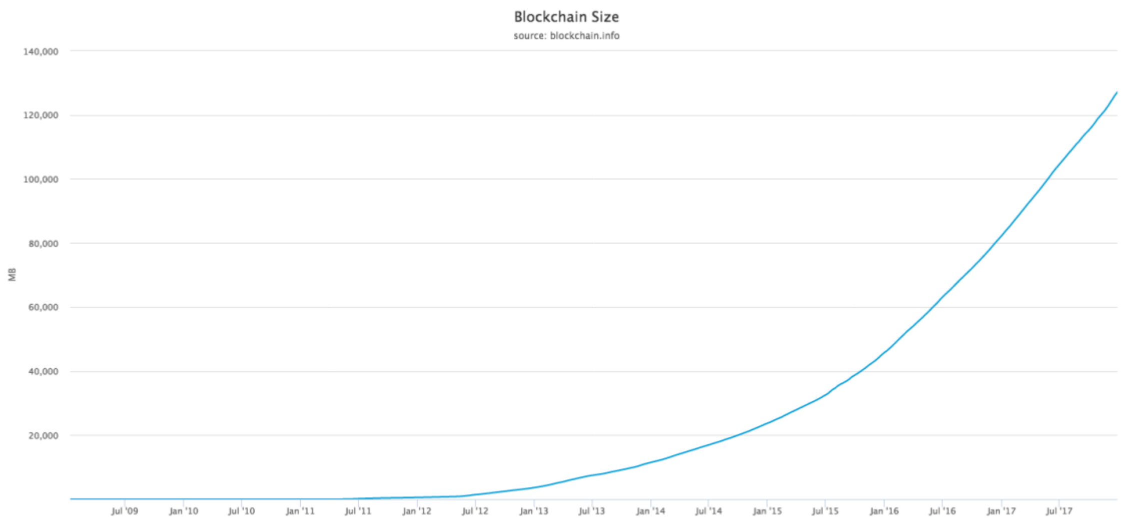
Bitcoin SPV Protocol

HingOn Miu
hmiu@andrew.cmu.edu

Background

Running a Bitcoin full node client to use its wallet to store Bitcoins is not always ideal for users, since full node clients have certain hardware requirements. According to bitcoin.org, full node requires over a hundred gigabytes of free disk space, 2 gigabytes of RAM, internet connection with upload speeds of at least 50 kilobytes per second, and 6 or more hours to be left running per day. So, it is fair to expect that most of that disk space is used to store the Bitcoin blockchain. As of 2017, according to statistics by blockchain.com, the total size of raw Bitcoin blockchain is about 130 gigabytes. Since full node client keeps a copy of entire raw blockchain and it could take hours or even days to download all blockchain files, casual Bitcoin user should not be always running full node client just to use its wallet to store a few Bitcoins.



According to original Bitcoin paper by Satoshi Nakamoto, Simplified Payment Verification (SPV) is a secure method for Bitcoin user to verify payments without running full node. Instead of keeping the entire blockchain like a full node client, SPV client only needs to keep a copy of all the block headers, as SPV protocol verifies a transaction by querying full nodes to obtain Merkle branches for that transaction. It then uses the Merkle tree structure for proof of inclusion by recomputing each ancestor hash with Merkle branches and matching the Merkle root hash in the block header, without the need to parse the entire block to look for that transaction.

As of 2017, according to statistics by blockchain.com, the total number of blocks in blockchain is about 500,000. As each block headers takes exactly 80 bytes, each instance of SPV client requires merely 500,000 * 80 bytes = 40 megabytes of disk storage for all block headers. Therefore, running a SPV client certainly has a much lighter hardware requirement than running a full node client, and so it is a better choice for casual Bitcoin users.

Usage

1. Install Python dependencies and Bitcoin core. Run Bitcoin full node to download complete raw .dat blockchain files.

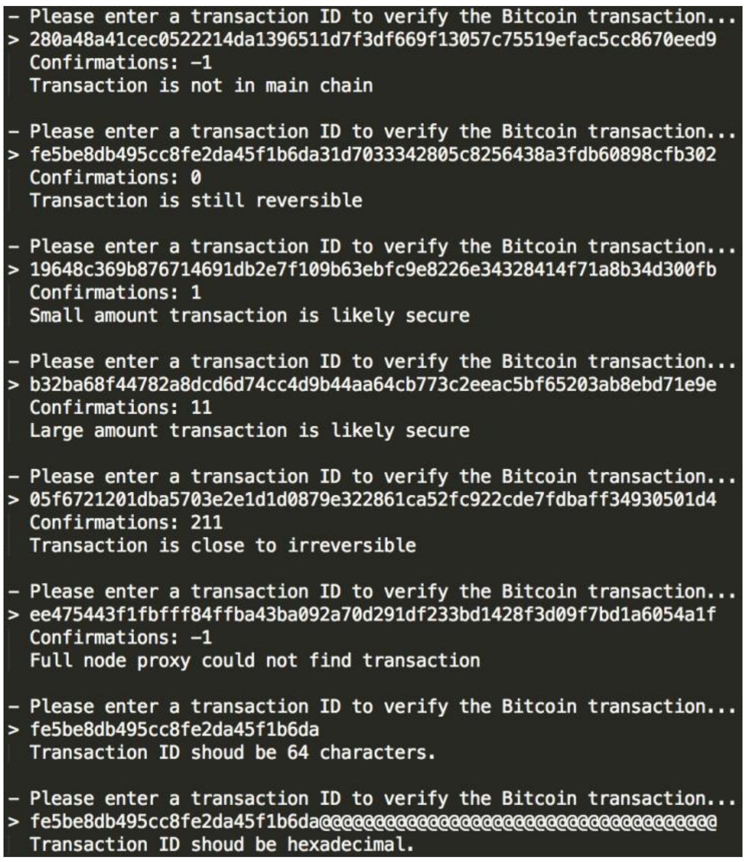
2. Run a local instance of full node proxy and provide the proxy the directory path

to the raw blockchain files. Wait for the full node proxy parses each block of the raw blockchain and stores all block headers to file.

3. Run remote instance(s) of SPV client. Wait for the SPV client to download and parse block headers generated by full node proxy.

4. Enter Bitcoin transaction hash (TXID) into SPV client interface to verify transactions and check confirmations.

Example



```
- Please enter a transaction ID to verify the Bitcoin transaction...
> 280a48a41cec0522214da1396511d7f3df669f13057c75519efac5cc8670eed9
  Confirmations: -1
  Transaction is not in main chain

- Please enter a transaction ID to verify the Bitcoin transaction...
> fe5be8db495cc8fe2da45f1b6da31d7033342805c8256438a3fdb60898cfb302
  Confirmations: 0
  Transaction is still reversible

- Please enter a transaction ID to verify the Bitcoin transaction...
> 19648c369b876714691db2e7f109b63ebfc9e8226e34328414f71a8b34d300fb
  Confirmations: 1
  Small amount transaction is likely secure

- Please enter a transaction ID to verify the Bitcoin transaction...
> b32ba68f44782a8dcd6d74cc4d9b44aa64cb773c2eeac5bf65203ab8ebd71e9e
  Confirmations: 11
  Large amount transaction is likely secure

- Please enter a transaction ID to verify the Bitcoin transaction...
> 05f6721201dba5703e2e1d1d0879e322861ca52fc922cde7fdbaff34930501d4
  Confirmations: 211
  Transaction is close to irreversible

- Please enter a transaction ID to verify the Bitcoin transaction...
> ee475443f1fbfff84ffba43ba092a70d291df233bd1428f3d09f7bd1a6054a1f
  Confirmations: -1
  Full node proxy could not find transaction

- Please enter a transaction ID to verify the Bitcoin transaction...
> fe5be8db495cc8fe2da45f1b6da
  Transaction ID shoud be 64 characters.

- Please enter a transaction ID to verify the Bitcoin transaction...
> fe5be8db495cc8fe2da45f1b6da@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  Transaction ID shoud be hexadecimal.
```

Full Node Proxy

Before running our proxy, we have to first run Bitcoin full node to retrieve the latest complete raw Bitcoin blockchain files. These files are named blkXXXXX.dat, and blk00000.dat is the first file of the raw blockchain. These files are usually located in ~/.bitcoin/blocks/. After downloading those raw blockchain files, we can start our full node proxy and run it localhost to listen for HTTP GET requests from SPV clients on port 9000 for testing purpose.

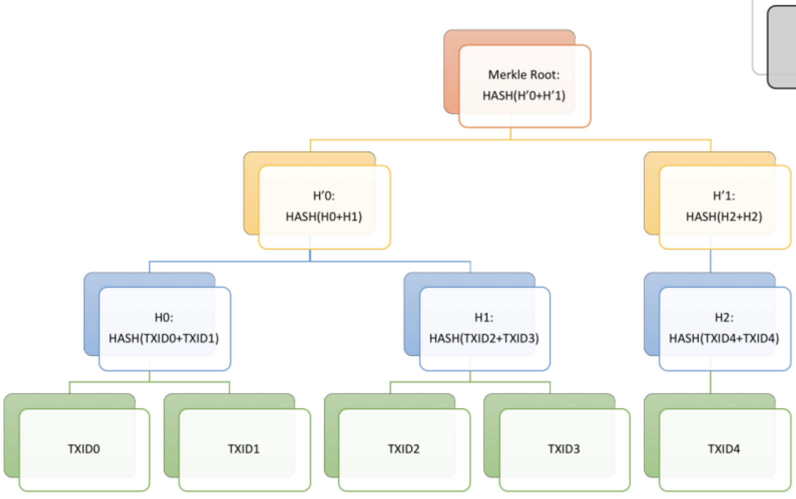
The absolute path of the local directory that holds those raw Bitcoin blockchain files has to be provided to our proxy. After starting our proxy, the setup stage is initiated before our proxy server forks worker threads to handle incoming HTTP connections. The setup stage loads and parses each blockchain file to compute block hash for each block and transaction hash for each transaction.

To compute the block hash of a block, which is the block header hash, our proxy parses version number, previous block hash, Merkle root hash, time, nBits and nonce of each block. These fields are concatenated to run SHA-256 hashing twice to compute the block hash. And of course, the block header (exactly 80 bytes) of each block is stored to file for SPV clients to retrieve later.

To compute the transaction hash of a transaction, our proxy server parses transaction version number, input transaction count, input transactions, output transaction count, output transactions and lock time of each transaction. These fields are concatenated to run SHA-256 hashing twice to compute the transaction hash. Our proxy also stores the index of each transaction within a block in a global transaction hash table so that finding transaction index of a transaction takes O(1) time. This transaction index is essential in constructing a Merkle tree efficiently, since it is the transaction's leaf index in Merkle tree.

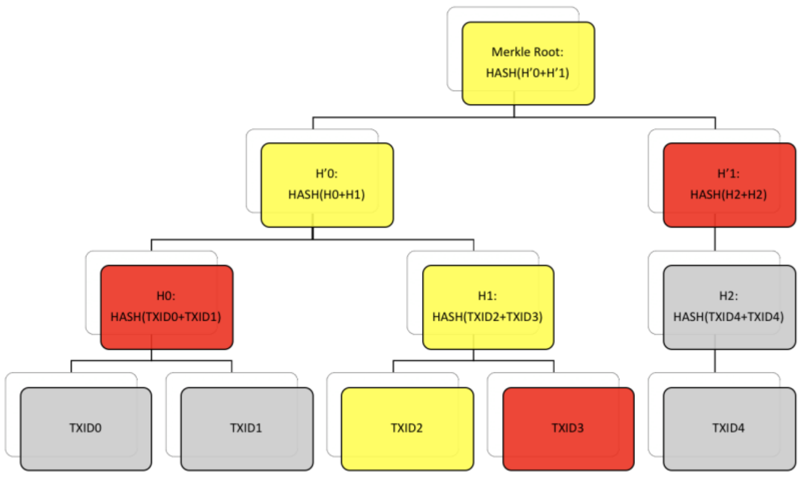
Merkle Tree Construction

For all transactions parsed in each block, our full node proxy runs Merkle tree hashing algorithm with all the computed transaction hashes to verify that the Merkle root hash in block header is correct. The bottom-up hashing algorithm runs on every level of the Merkle tree from the bottom leaf hashes. Each pair of child hashes are concatenated to run SHA-256 hashing twice to compute the parent hash. The concatenation order of each child hash pair is determined by index of the child hashes in each tree level. If a level of Merkle tree has an even number of hashes, each pair of hashes is concatenated and hashed in index order. If a level of Merkle tree has an odd number of hashes, the last hash is replicated and appended to the end. The algorithm terminates only when a single hash is left in the tree, which is the resulted Merkle root hash. Below diagram illustrates a case of computing Merkle root hash for a total of 5 transactions in a block.



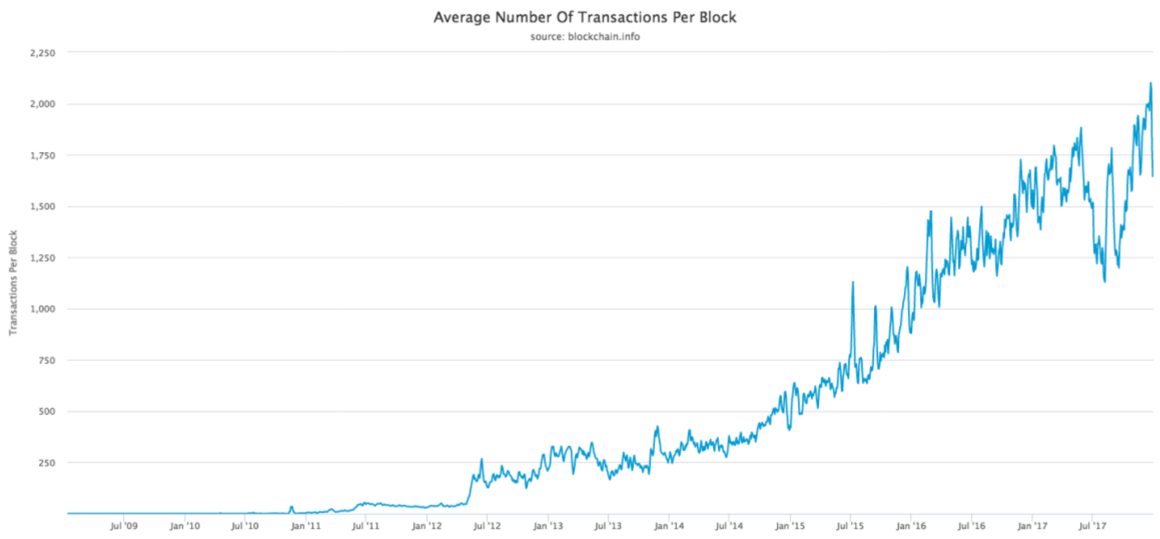
The entire Merkle tree of each block is cached so that full node proxy can respond Merkle branches to SPV client's transaction query faster. Our proxy stores the cached Merkle tree for each block in a global block hash table so that finding cached Merkle tree of a transaction takes O(1) time. Therefore, as a SPV client query our proxy server a transaction hash (TXID), our proxy first finds its transaction index in O(1) time using the global transaction hash table. Next, our proxy server finds the transaction's corresponding cached Merkle tree in O(1) time using the global block hash table.

With the cached Merkle tree and the leaf index of the transaction, our proxy server traverses the Merkle tree bottom-up level by level to collect the transaction's Merkle branch pair in each level. Below diagram illustrates a case of collecting Merkle branches (in red) for leaf TXID2 to compute the Merkle root hash.



Obviously, bottom-up traversal of Merkle tree to collect Merkle branches takes O(log h) time, where h is the number of transactions within a block. Since finding a transaction index takes O(1) time and collecting Merkle branches in a cached Merkle tree takes O(log h) time, the overall response time of our full node proxy to answer SPV client's transaction query is O(log h). Without the Merkle tree cache, constructing the whole tree of a block to collect Merkle branches for each SPV client transaction query takes O(h * log h) time. This is considerably slower than caching Merkle tree in full node proxy's setup stage.

Now, let's look at the memory trade-off for caching Merkle tree in setup stage. Since the size of a Merkle tree takes (h * log h) / 2 space, the memory of caching Merkle tree for a block is then O(h * log h). Therefore, the memory of global block hash table to cache Merkle tree for all blocks is then O(b * h * log h), where b is the total number of blocks in blockchain. As of 2017, according to statistics by blockchain.com, the average number of transaction per block is around 1500, and the total number of blocks in blockchain is about 500,000. As each Merkle branch is essentially a 64-



character long SHA-256 hexadecimal hash, each Python Merkle branch hash string takes about 100 bytes (check Python built-in type sizes with sys.getsizeof). Therefore, caching Merkle tree for all blocks takes 500,000 * (1500 * log 1500) / 2 * 100 bytes ≅ 120 gigabytes.

Certainly, a practical approach is caching only recent 3-day or 1-week blocks' Merkle tree as it becomes unlikely for SPV client users to frequently verify very old transactions. So, some response delay for proxy server to construct complete Merkle tree to collect Merkle branches for old transactions is likely to be acceptable to users. As of 2017, according to statistics by blockchain.com, the average number of blocks mined per day is about 150. So, caching Merkle tree for 3-day amount of blocks in proxy setup stage takes about 3 * 150 * (1500 * log 1500) / 2 * 100 bytes ≅ 100 megabytes. With this in mind, we can more fairly decide the response time and memory trade-off for our full node proxy.

SPV Client

After our full node proxy writes all block headers to file, we can run multiple remote instances of SPV client. After starting our SPV client, the setup stage is initiated to download all block headers from full node proxy. Next, our SPV client parses each block header to construct a global header structure hash table, in which each block header's previous block hash is the key. Therefore, it is now possible to traverse the blockchain starting from the genesis block header, as our SPV client can compute next block header's block hash from the block header stored in the hash table. Note that the previous block hash of the genesis block is all zero bytes, and so our client can easily fetch the genesis block header from hash table with zero string hash.

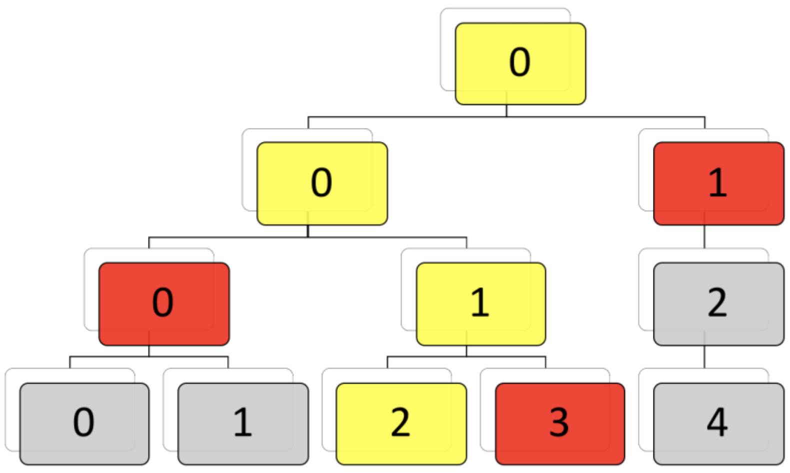
To compute the block hash of a block header, our SPV client parses version number, previous block hash, Merkle root hash, time, nBits and nonce. These fields are concatenated to run SHA-256 hashing twice to compute the block hash. With the block hash of each header, our SPV client can identify next block header using the header hash table.

Next, our SPV client runs breadth first search from the genesis block header to compute the distance of each block header from genesis block header. So, knowing the distance of each block header, our SPV client can identify the longest path from genesis block header, which is the main chain of blockchain. This breadth first search pre-processing computation in setup stage has O(V + E) runtime, so that our SPV client can answer user's transaction queries to verify whether a transaction belongs to a block in main chain with O(1) runtime.

Merkle Tree Verification

When SPV client completes setup stage, user can interact with the client interface to enter transaction hash (TXID) to verify Bitcoin transactions and check confirmations. After a transaction hash is inputted, our SPV client makes HTTP query of this transaction to full node proxy to get the transaction's corresponding block's transaction count, transaction index within the block and the Merkle branch pairs of every tree level.

After full node proxy responded, our SPV client bottom-up constructs the Merkle tree structure of the block using just the transaction count (the number of Merkle leaves at bottom tree level) and the transaction index (the index of the Merkle leaf). With the Merkle tree structure, our SPV client now identifies the hashing order with each Merkle branch in each level. The hashing order is whether concatenating to right or left of the Merkle branch pair for each tree level before SHA-256 hashing. This bottom-up Merkle tree structure reconstruction takes O(log h) time, where h is the number of transactions within a block. Below diagram illustrates a case of determining the hashing order of each Merkle branch pair with each level's indexes in Merkle tree structure.



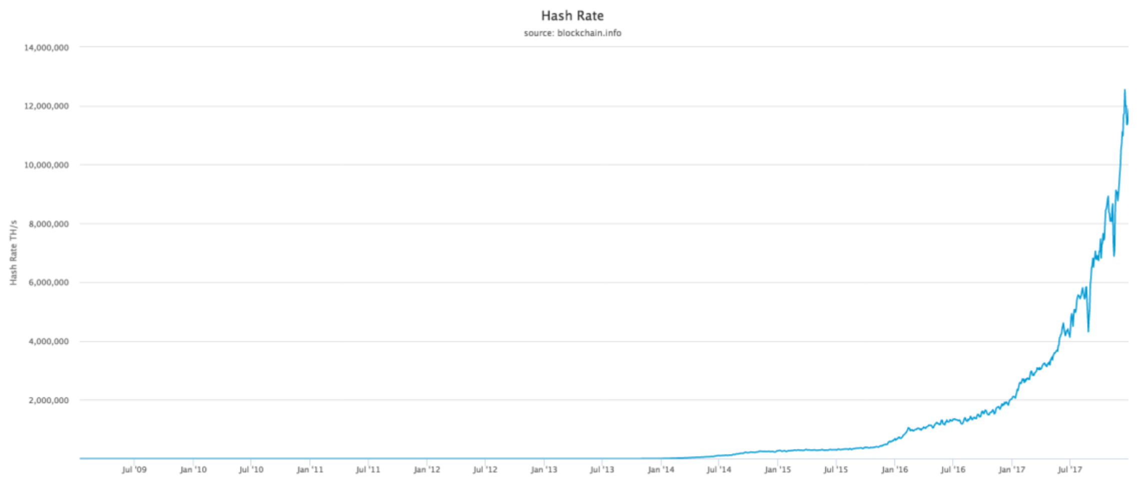
With the hashing order of each Merkle branch pair determined, our SPV client bottom-up computes the Merkle tree root using the Merkle branch pairs retrieved from full node proxy and the Merkle leaf, which is the user's transaction ID to verify. If the computed Merkle tree root matches the Merkle root hash initially parsed in block header during setup stage, our SPV client then verifies the user's transaction ID indeed belongs to that block. This Merkle branches verification takes O(log h) time, as bottom-up computing Merkle root with Merkle branches traverses level by level.

Confirmations

Once the Merkle tree verification using Merkle branches proves the existence of the user's transaction in the block, our SPV client then computes block depth to evaluate transaction's validity and security. Since the number of confirmations of a transaction is just the block depth of the block, our SPV client can quickly compute it in O(1) time using the initially computed block height during the setup stage. Now, our SPV client responds the user to verify whether the transaction is embedded in main chain as well as the number of confirmations of this transaction.

To defend against double spending attack, a Bitcoin transaction should be at least 6 blocks deep, which means getting at least 6 confirmations. Of course, the deeper a transaction is buried in main chain, the harder for an attacker to be able to manipulate it. Getting at least 6 confirmations to confirm a transaction is widely adopted by many popular Bitcoin clients. This is based on the assumption that an attacker is very unlikely to amass more than 10% of the total hash rate of the Bitcoin network. As of 2017, according to statistics by blockchain.com, the total hash rate of Bitcoin network is about 10,000,000 tera hashes per second, and so 10% of total computation power of Bitcoin network is about 1,000,000 tera hashes per second.

In comparison, the most powerful Bitcoin miner in 2017 market is AntMiner s9. It costs about 2000 USD and it has a



hash rate of about 10 tera hashes per second. In other words, the attacker should have control over the computation power of about 100,000 AntMiner s9 to manipulate a transaction with 6 confirmations. These Bitcoin miners together would cost about 200 millions USD, not to mention electricity fees and other hardware equipment like power supply units and cooling solutions to run these Bitcoin miners.

Thus, it is generally good practice for SPV client users to wait for 6 or more confirmations for each Bitcoin transaction, because it becomes exponentially more expensive to reverse a transaction. And for large amount transactions, users should use full node Bitcoin client instead of SPV client, since full node client would verify every block in blockchain.

Conclusion

Merkle tree is the key in SPV client verifying Bitcoin transaction. Some may wonder why Merkle tree scheme is chosen to hash all transactions within each block instead of simply hashing all transactions in order sequentially. It is true that sequential hashing scheme works, since manipulating any one of the transactions within the block would also change the final computed hash. However, in order for remote thin client to verify a transaction, full node would need to send over all other transactions within the block for thin client to verify the final hash in block header. Therefore, the network bandwidth usage is O(h) for each transaction verification, where h is the number of transactions within a block.

While a tree structure hashing scheme like Merkle tree only requires the number of tree levels of computations for any transaction within the block to get the final root hash. So, full node would need to send over the number of tree levels of hash branches for remote thin client to verify the final root hash in block header. Certainly, manipulating any one of the transactions within the block would change the final computed Merkle root hash. Therefore, the network bandwidth usage of our SPV client is O(log h) for each transaction verification.

As of 2017, according to statistics by blockchain.com, the average number of transaction per block is around 1500. To put into perspective, thin client using sequential hashing scheme would then require all other 1499 transaction hashes from full node to verify a transaction. In comparison, Merkle tree scheme would have 1500 leaves, 750 hashes in next level, then 375 hashes, 188 hashes, 94 hashes, 47 hashes, 24 hashes, 12 hashes, 6 hashes, 3 hashes, 2 hashes and the final Merkle root hash. There is a total of 12 tree levels, and so SPV client using Merkle tree scheme would only require 12 transaction hashes from full node to verify a transaction. Clearly, SPV protocol on average uses much less network bandwidth to verify transactions.

To sum up, our full node proxy takes O(log h) time to collect Merkle branches using cached Merkle tree to respond our SPV client's transaction query. Then, our SPV client also takes O(log h) time to compute Merkle root using the Merkle tree branches to verify the user's transaction ID. Therefore, the complete Bitcoin transaction verification service of our SPV client only takes O(log h) time and O(log h) network bandwidth.

Sources

https://blockchain.com/
https://bitcoin.org/
https://bitcoin.stackexchange.com/
https://en.bitcoin.it/
https://bitcoinmining.com/