Distributed Robust Accounting

Applied Cryptography Project

1. Motivation

Bitcoin is an experimental, decentralized digital currency that enables instant payments to anyone, anywhere in the world. Bitcoin uses peer-to-peer technology to operate with no central authority: managing transactions and issuing money are carried out collectively by the network. Bitcoin is designed around the idea of using cryptography to control the creation and transfer of money, rather than relying on central authorities.

However, Bitcoin requires that every node stores the entire ledger and broadcasts every transaction to every node. This limits the scalability of the system and also constitutes an entry barrier: it takes many hours for a new installation to download the entire block chain (which is how the transaction ledger is called in Bitcoin terminology). Except for the most expensive models, cellphones do not have the storage capacity to store all the transactions and it is also often prohibitively expensive in terms of mobile communication to maintain a bitcoin network node on a cellphone. At present, this implies that mobile devices and other low-power computers must trust one or more nodes on the bitcoin network which is slowly but surely turning into a centralized structure quite out of line with its original ethos.

However, Bitcoin's design includes several features that seem to be aimed at decentralization, allowing nodes to hold only partial copies of the block chain and still not trust other nodes for verifying its integrity regarding transactions and accounts of interest. The goal of this project is to design and maybe prototype such decentralization.

2. Background

Firstly, let's recap how a regular full node works. The fundamental problem Bitcoin solves is achieving consensus on who owns what. Every node maintains a database of unspent outputs, and transactions that attempt to spend outputs that don't exist or were already spent are ignored. Blocks are solved by miners and broadcast to ensure everyone agrees on the ordering of transactions, and so nodes that don't see a broadcast transaction for some reason (eg, they were offline at the time) can catch up.

The act of checking, storing and updating the database for every single transaction is quite intensive. Catching up to the current state of the database from scratch is also very slow. For this reason, not every computer can run a full node.

Transactions in each block of the block chain are hashed in a so-called Merkle tree, which can be used to verify any transaction without having to download all. From the vocabulary page of Bitcoin wiki:

Merkle root Every transaction has a hash associated with it. In a block, all of the transaction hashes in the block are themselves hashed (sometimes several times - the exact process is complex), and the result is the Merkle root. In other words, the Merkle root is the hash of all the hashes of all the transactions in the block. The Merkle root is included in the block header. With this scheme, it is possible to securely verify that a transaction has been accepted by the network (and get the number of confirmations) by downloading just the tiny block headers and Merkle tree - downloading the entire block chain is unnecessary. This feature is currently not used in Bitcoin, but it will be in the future.

Each node is particularly and primarily interested in transactions that concern the accounts that are controlled by their operators, perhaps some watched accounts (such as those of counterparties). It needs to verify that all the transactions preceding them beginning with emission (mining) have been arithmetically correct and that no double spending has been attempted. It may also verify random transactions as a means of spot-checking the network.

3. Solution

The solution is a simplified Bitcoin client that doesn't store all the transaction and the blockchain, only transactions that are relevant to the wallet are stored. Every other transaction is thrown away or simply never downloaded. The block chain is still used and broadcast transactions are still received, but those transactions are not and cannot be checked to ensure they are valid. This mode of operation is fast and lightweight enough to be run on a smartphone.

3.1.

Each balance is simply associated with an address and its public-private key pair. The money "belongs" to anyone who has the private key and can sign transactions with it. Moreover, those keys do not have to be registered anywhere in advance, as they are only used when required for a transaction. Each person can have many such addresses, each with its own balance, which makes it very difficult to know which person owns what amount. In order to protect his privacy, Bob can generate a new public-private key pair for each individual receiving transaction and the Bitcoin software encourages this behavior by default.

When Bob someone starts to use this simplified wallet, he generates new public-private key pairs, so the wallet knows the creation time of all its keys. To keep track of its future transactions, block contents before this time don't have to be downloaded, only the headers, so it's much faster to bootstrap the system in this way. The method is similar to bitcoinj's fast catchup. This way you can sync with the chain just by downloading headers and some transactions+Merkle branches, but sometimes this is still too slow.

It can further be augmented with bitcoinj's chekpoint files. These are generated using the BuildCheckpoints tool that can be found in the tools module of the bitcoinj source code. BuildCheckpoints downloads headers and writes out a subset of them to a file. That file can then be shipped with your application. When you create a new BlockStore object, you can use that file to initialise it to whichever checkpointed block comes just before your wallets fast catchup time (i.e. the birthday of the oldest key in your wallet). Then you only need to download headers from that point onwards.

Checkpoints are called checkpoints because, like the upstream Satoshi client, once you've initialized the block store with one bitcoinj will refuse to re-organise (process chain splits) past that point. In fact, it won't even recognise that a re-org has taken place because the earlier blocks don't exist in the block store, thus the alternative fork of the chain will be seen merely as a set of orphan blocks. For this reason the BuildCheckpoints tool won't add any checkpoints fresher than one month from when it's run - it only takes a few seconds to download the last months worth of chain headers, and no fork is likely to ever be longer than one month.

When a transaction is broadcast over the network we say it is pending inclusion in a block. Mining nodes will see the transaction, check it for themselves and if it's valid, include it in the current block they're trying to solve. Nodes do not relay invalid transactions. Your app will receive pending transactions, add them to the wallet, and run event listeners.

Bloom filtering By default the PeerGroup and Wallet will work together to calculate and upload Bloom filters to each connected peer. A Bloom filter is a compact, privacy preserving representation of the keys/addresses in a wallet. When one is passed to a remote peer, it changes its behaviour. Instead of relaying all broadcast transactions and the full contents of blocks, it matches each transaction it sees against the filter. If the filter matches, that transaction is sent to your app, otherwise it's ignored. When a transaction is being sent to you because it's in a block, it comes with a Merkle branch that mathematically proves the transaction was included in that block. BitcoinJ checks the Merkle branch for each transaction, and rejects any attempts to defraud you.

Bloom filters can be noisy. A noisy filter is one that matches more keys or addresses than are actually in your wallet. Noise is intentional and serves to protect your wallet privacy - a remote node can't know if a matched transaction is really yours or not. In theory, wallet keys/addresses could be split up across each connected node for even more privacy. Essentially it's a bandwidth vs privacy tradeoff - a higher FP rate confuses remote eavesdroppers more, but you have to download more useless data as a result.

3.2.

The solution is based on distributed hash tables. Nodes stores their own transaction plus some other transaction.

A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table; (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

Key space partitioning:

Each node has an id. It stores the transactions wich ?public key? is closest to this id

szóval itt ki kéne találni h mi legyen az id-ja a node-oknak és hogy a tranzakciókat mi alapján rendeljük hozzá. pl lehetne a tranzakciót a hash-e alapján hozzárendelni egy node-hoz- ez esetben akkor a node id-ja is egy random 32 bites szám kéne hogy legyen. itt a következő a problémám: ha valaki megfigyeli egy node forgalmát (h milyen tranzakciókat tárol el) akkor mivel nem csak a sajátját tárolja el nem tudja h mik az ő tranzakciói. de a tarnzakciók azonosítójából ki tudja szűrni h azt a node azért tárolta-e el mert közel van az id-jéhez vagy mert az övé a tranzakció. erre ugye megoldás lenne h a node nem hozza nyilvánosságra az id-jét, de ekkor a későbbiekben leírt routingot nem lehetne megvalósítani...

easy to add new nodes

az id alapján az új node-ok tudni fogják h milyen tranzakciókat tárolјanak, és a broadcastolják az id-jukat amikor belépnek a networkbe, akkor a többi node is tudni fogja h ha van esetleg egy tranzakció amit korábban ő tárolt volna, az most már az új node-hoz közelebb van. de ha egy tranzakiót ketten is eltárolnak abból sincs semmi baj

easy to find a transaction (send message to a node whose id is closer to the transactions key than yours and so on, and it comes back the same way. here we need some proper routing protocol ->crypto prot)

szóval itt egy routing protocol-lal utánaérdeklődik egy adott node egy tranzakciónak. ha nem is tudja minden másik node id-ját, elküld egy requset üzenetet egy node-nek akinek az id-ja közelebb van a tranzakcióhoz, és az megint továbbítja egy olyannak aki még közelebb van, mígnem el nem ér ahhoz akinél van a message. itt ugye szükséges h a node-ok egy összefüggő gráfot alkossanak (az élek azt jelölöik akinek az id-ját ismered), erre akkor kell valami feltétel h hány szomszédja legyen egynek. vagy valami ilyesmi. az is kérdéses, hogy itt mi alapján definiálunk egy szomszédot. mivel broadcast communication van, logikus lenne h a hozzá közeliek a szomszédok, de mivel ezt az alkalmazást pl smartphonokra cináljuk, azoknak a helyzete változik.

all transaction are stored in such a decentraliezd network, nodes store their own transaction plus some transactions that are close to them, if they need data of a transaction they send routing messages to other nodes that can sore that specific transaction. each node stores the block headers

when broadcasting a transaction a node stores the ones that belong to him or that are close to him. when broadcasting a block each node checks the transactions in it and stores merkel branches of the previously stored transactions

szerintem ha mindenki eltárolja az összes block header-t és az összes tranzakció valahol a networkben el van mentve, akkor ezzel tudnak verifikálni mindent, és megvalósul a decentralizáltság. ellenvetéseket várok.. :)

plusz még kérdéses: DHT-t használunk, de akkor ehhez kell a bloom filter? valahogy a bloom filter fogja egvalósítani ezt a dht-s elrendezést? végülis ha a bloom filter minden tranzakcióhoz hozzárendel egyértelműen valami értéket, és azt nézi meg h közel van-e a node-hoz, akkor egy attacker a tranzakciók hash-ét hiába nézi, nem tudja h a node csak azért választotta mert közeli vagy mert az övé.

3.3.

First step: the DHT method works only if there's a sufficiently large network of these simplified clients. Until that time the clients will work as follows: from the creation of the wallet they check every broadcast message wheter it contains a transaction they are interested in. If there's no such transaction, they drop the message. After a transaction they're interested in is broadcasted, they start to check the generated blocks wheter it cointains taht transaction or not. With this method they can verify a transaction. If a transaction is stored in a block the simplified wallet stores the corresponding merkel branch.

Nodes can store some other transactions as well so that attackers won't know which transactions belong to them.

Nos ezt nem tudom h mennyire működik így jól, de lényegében a bitcoinj is ezt csinálja, szóval szerintem elfogadható

3.4.

Further problems: avoid double spending, attacks, level of trust in other nodes

ezeket a dolgokat át kéne gondolni, hogy miért nem lehetséges duoble spend, milyen támadások lehetnek, blablabla...

Hivatkozások

[1] https://en.bitcoin.it/wiki/Main_Page

[2] http://en.wikipedia.org/wiki/Distributed_hash_table

[3] https://code.google.com/p/bitcoinj/