

Purely P2P Crypto-Currency With Finite Mini-Blockchain

J.D. Bruce
May 2013. Rev 1.
www.bitfreak.info

Abstract

Almost all P2P crypto-currencies prevent double spending and similar such attacks with a bulky “blockchain” scheme, and the ones which do not typically use some sort of pseudo-centralized solution to manage the transactions. Here I propose a purely P2P crypto-currency scheme with a finite blockchain, dubbed the “*mini-blockchain*”. Each time a new block is solved the oldest block is trimmed from the end of the mini-blockchain so that it always has the same number of blocks. It is argued that the loss of security this trimming process incurs can be solved with a small “*proof chain*” and the loss of coin ownership data is solved with a database which holds the balance of all non-empty addresses, dubbed the “*account tree*”. The proof chain secures the mini-blockchain and the mini-blockchain secures the account tree. This paper will describe the way in which these three mechanisms can work together to form a system which provides a high level of integrity and security, yet is much slimmer than all other purely P2P currencies. It also offers other potential benefits such as faster transactions and lower fees, quicker network synchronization, support for high levels of traffic, more block space for custom messages, and increased anonymity.

Introduction

It was over four years ago now that “Satoshi Nakamoto” first released Bitcoin into the public domain and forever changed the way many people think about finance and economics. [1] Back then the difficulty was low and the blockchain was small. For the first two years things looked great, many believed the blockchain wouldn't become problematic for a long time so the issue was put to the side. We are now in May of 2013 and the blockchain is close to 7 gigabytes in size. [2] While still manageable, it is becoming a serious issue of concern.

The core developers of Bitcoin now direct much of their attention to handling the ever-increasing levels of network traffic. Today the bitcointalk.org forum observes almost daily threads concerning methods for minimizing the size of the blockchain and decreasing the synchronization time. One of the most effective measures taken to date was a switch from Berkeley DB to LevelDB. [3] BDB is much slower and the switch over to LDB resulted in a major performance boost in terms of synchronization and block verification speeds.

Another promising endeavor is the “Ultimate blockchain compression” project [4] which aims to achieve “near-optimal blockchain compression” by implementing blockchain pruning techniques including “balance-tree information” which is “maintained and verified in a separate blockchain through merged mining”. Blockchain pruning is indeed a promising venture and could prove to provide a high level of compression, but this particular proposal is extremely complex and now it looks like there might be a flaw in the idea.

The bitcointalk forum also observes many threads about changing the max block size limit (with many for and against it). There is now a group of people creating a “Bitcoin Blocksize Problem Video” [5] in which they argue against increasing the max block size because “it will become more expensive to run a node”. Gavin Andresen, Lead Core Bitcoin Developer, replies “The block size will be raised. Your video will just make a lot of people worried about nothing”. Every day this issue seems to become more pressing.

The max block size definitely will be raised at some point, right now the transaction capacity is restricted to 7 transactions per second [6] and eventually that just won't be enough. Raising the max block size in Bitcoin will require a coordinated hardfork because all the older clients won't know how to deal with the larger blocks. So there are legitimate reasons to be concerned about increasing the max block size, but that doesn't mean it's not needed. There are several cons and pros which must be considered.

Increasing the max block size would increase transaction bandwidth and lower fees, but it would cause the blockchain to grow even faster and put more stress on nodes. Many members of the bitcointalk forum say it's not the max block size which is a problem, instead they blame gambling services such as Satoshi Dice for “spamming” the network with many transactions. Other members note that Satoshi Dice “is providing stress-testing for the network, and showing that unless the block size limit is lifted, there will be a problem”. [5]

Should we be Concerned?

As mentioned a moment ago, there are several ways we can attempt to deal with the size of the blockchain. On the Bitcoin Stack Exchange a user Sean Chapman asks “Are there any studies into the size of the blockchain scaling over time?”. Meni Rosenfeld, writer of the top answer, explains that although he isn't aware of any studies that meet Chapman's request, he can outline 5 reasons why we needn't be concerned about blockchain scalability [7]:

1. Not every user needs to run a full network node.
2. Spent outputs can be pruned from the blockchain.
3. Off-chain transactions can help reduce stress on the network.
4. Transaction fees can offset the block storage costs.
5. Moore's Law is still going strong for the foreseeable future.

Satoshi spoke about the 1st point back in late 2008 when he said as the network grows mining “would be left more and more to specialists with server farms of specialized hardware” [8]. This is perhaps a satisfactory solution but it will result in continued centralization and eventually these specialists will control a large percentage of the network power by making it harder for the smaller players to participate. The 2nd point was already mentioned; pruning offers promise but it's a complex process and the results are limited.

The main problem with just pruning spent outputs is that a lot of the “dust” generated by services such as Satoshi Dice still clog up the system. One proposal on the bitcointalk forum suggests a method for solving this. [9] If spent outputs can be pruned efficiently and if all the dust can be “swept up” by miners using a special “refresh transaction” it may be possible to create a “rolling blockchain” because we can prune off all the unnecessary blocks and pack everything into a dynamically sized rolling blockchain.

This proposal really would kill many birds with one stone, but it is not as simple as it sounds and it has some stumbling blocks which need to be figured out. For example it's difficult to determine how miners are rewarded for sweeping up dust unless we expect some miners to simply do it for free because if they help to trim down the blockchain it will reduce their server costs, therefore providing incentive for miners to not only put energy into sweeping up dust but also to include those special refresh transactions in their blocks.

The last 3 points mentioned by Rosenfeld all have some validity to them as well, but regardless of those points we are still stuck with a blockchain which never stops growing and it still doesn't provide us with a truly light weight scheme. Bitcoin probably has a long future ahead of it, and I think many of those 5 points will play a big role in whether it succeeds or not. There probably isn't a reason to be overly concerned just yet. In another 5 or 10 years we will have a much better idea about how well Bitcoin can scale.

Odds are that Bitcoin is here to stay for a while, and even if the scheme described in this paper is verified and implemented, it won't make Bitcoin suddenly obsolete. It will offer an alternative with better functionality in some areas and poorer functionality in other areas. But at the end of the day we shouldn't play favorites, we should try to build on the work of Satoshi and create an open free market of the best crypto-currencies where competition can thrive. The scheme proposed here can provide us with truly fresh and unique advantages.

We should be looking for innovative new ways to solve these scalability issues in a concise and satisfying manner. The Bitcoin Wiki states "At very high transaction rates each block can be over half a gigabyte in size" [6]. Is it really feasible for Bitcoin to reach that level of network traffic and attempt to store it all in an ever-growing blockchain? Perhaps if something like the "rolling chain" idea is implemented properly it could work, but beyond that seems little hope. Coincidentally, the scheme proposed here does use something which might be called a "rolling chain".

Finding a Solution

Most of the pruning proposals for Bitcoin are very complex and require a large change to the way the blockchain works, making them very difficult to implement. This proposal is not an attempt to change Bitcoin in any drastic way, it is a proposal for an entirely new alternative crypto-currency which eliminates the need for a full blockchain by replacing it with other light weight mechanisms. This is a problem many people have set out to solve but it seems we still lack any satisfactory solution. Until now perhaps.

The proposed solution (untested) described in this paper comes by understanding the different purposes of the blockchain and then separating that functionality into individual mechanisms which are each optimized to serve their purpose. The blockchain has 3 main functions. The Bitcoin blockchain combines these functions into one single mechanism and as a result doesn't scale well. It requires you to store a lot of data which doesn't really need to be stored forever. Breaking up the functions of the blockchain is the key.

Functions of the Blockchain:

1. to coordinate how the network processes transactions
2. to encapsulate the proof-of-work which secures the network
3. to manage account balances; record the ownership of coins

The original flawed version of this concept was proposed on the bitcointalk forum in March of 2013. [10] The concept was summarized and slightly expanded upon in April of 2013. [11] With help from other bitcointalk forum members, we were able to fine tune the concept into something workable, but the idea still had one major problem. The scheme originally disregarded the need for a blockchain and relied upon the “address database”, now renamed the “account tree”. We needed a good way to secure the account tree without a blockchain.

We didn't want a full blockchain, but maybe a finite “mini-blockchain” could provide the answer. After some discussion and a lot of thinking it became clear the mini-blockchain could act as a reasonable solution but it wasn't something I could bet my life savings on. It had weak points which, although difficult to exploit, if exploited one could destroy the entire system single-handedly, and such a scheme is simply not adequate when dealing with wealth. I needed a more absolute level of security before I could be satisfied.

The last piece of the puzzle finally came to me near the end of April 2013. The solution I have dubbed the “proof chain”. The proof chain helps to secure the mini-blockchain by encapsulating the work produced by the network, while remaining tiny in size. The account tree is only as secure as the mini-blockchain. This is a very simple generalization of the scheme. As we move through the following sections of this paper it will become clear how these components function, how they fit together, and what their purpose is.

The Account Tree

This proposal starts with the concept of an “account tree”. Why should we record every single transaction and save it forever if all we need to know is the balance of all non-empty addresses? The 3rd function of the blockchain is replaced with the account tree. The account tree will contain every unique non-empty address (or public key) and the balance of all those addresses, along with another value called the “Ref Block” which we will look at momentarily. It could also contain some other data but it needs to be as slim as possible.

When the balance of an address changes all we need to do is update numbers in the database instead of adding new data to the database. Of course this won't provide a truly finite amount of data to work with because new non-empty addresses will be appearing all the time, but it comes as close to finite as is probably possible. It is finite in some sense because the coins will have limited divisibility, and we can't really expect the world population or the number of internet users to continue growing forever. In any case it's scalable and manageable.

Even with a population of 10 billion people where each person had 10 different non-empty addresses, we would only need to keep track of 100 billion addresses. Since we can remove empty addresses from the database and since a transaction would simply require peers to shift around numbers in this database instead of adding new data to it, the size of the account tree should always remain considerably small. By the time we reach anything close to 100 billion unique non-empty addresses our computers will be much faster.

Owners of a non-empty address in the account tree would prove their ownership with their private key. Like Bitcoin, transactions would be created as a signed set of data and broadcast over the network. Like Bitcoin, miners who accept the transaction would then put it into their blocks and work on solving a difficult problem to get it into the mini-blockchain (more on this in the next sections). Nodes who accept the block would update their own copy of the account tree by shifting around coins or doing what ever was necessary.

The proposed database is named the “account tree” because it should have a Merkle tree structure. Each “account” in the tree has a corresponding hash. We group the accounts into “sectors” of something like 1000 accounts to each sector. We use the account hashes to calculate our “sector” hashes. Finally, we use all the sector hashes to calculate the master hash. Note that an “account” isn't a collection of addresses like a “Bitcoin account”. In this case each account refers to only one address (obviously normal “accounts” will also exist).

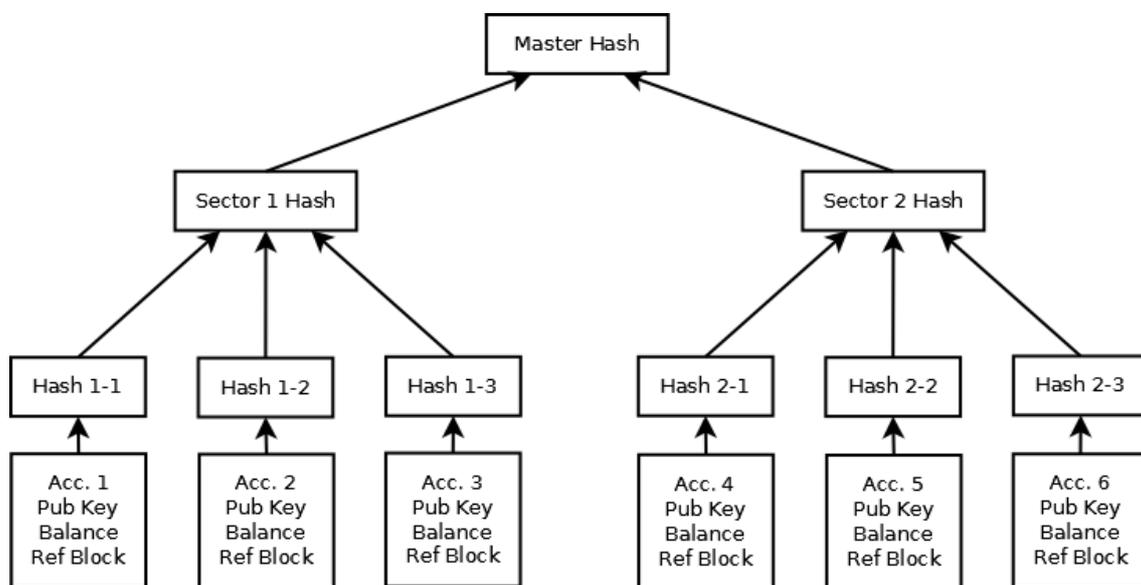


Figure 0-a

Figure 0-a shows a simplified account tree structure with only 2 sectors which each link to 3 unique accounts. A sector hash will change even if just one account within that sector is altered in any way. All the sector hashes together produce what has been dubbed the “master hash”. Any time anything in the account tree changes the master hash will change to reflect that change. This hash tree system provides consistency and integrity to our account data because the master hash is also stored in block headers (more on this shortly).

Each account holds the 3 fields mentioned previously, the public key, the balance, and the ref block. “Ref block” means “reference block” and its value is the number of the block in which the account lasted acted as an input. Or put another way, it's the number of the block in which the owner of the account last signed a transaction using that account. The ref block field is there to ensure the same signed transaction can't be used twice. We will discuss inputs and outputs in a later section so don't worry too much about that right now.

The proposed hash tree scheme allows nodes who have been offline longer than the total cycle time of the mini-blockchain to quickly discover which sectors of the account tree have changed in their absence. However, without a way to coordinate when and how the account

tree is changed we still can't ensure consistency between nodes. That is where the mini-blockchain comes in. Each time a new block is accepted into the mini-blockchain, nodes will use that block to update their copy of the account tree.

The Mini-Blockchain

The mini-blockchain provides our 1st blockchain function. The mini-blockchain is very similar to the Bitcoin blockchain, however it is finite in nature because as new blocks are solved and added to the front of the mini-blockchain old blocks are trimmed off the end. Well again, it isn't truly finite, because changing the max block size could increase the average size of the mini-blockchain. Later in this paper I do propose a mechanism for having a dynamically determined max block size but it's not a necessary part of the scheme.

If we are going to keep track of our database with a set of sector hashes and a master hash, we can't allow every single separate transaction to alter the database on demand. We must break them up into groups of transactions which are inserted into the database in periodic intervals of time. Without having the transactions solved in groups of transactions as blocks we have no viable method of maintaining the account tree. This creates an inherent need for something like a blockchain, but we can settle with a mini-blockchain.

Bitcoin requires the full blockchain because that's the only real way to determine the full balance of all the addresses. However we have the account tree to fulfill the task of managing account balances and recording the ownership of coins. We don't need the full thing, we can throw away old blocks and save an immense amount of disk space. However we do keep a few hundred or a few thousand of the newest blocks and that makes our mini-blockchain. The mini-blockchain also provides us with a level of security.

Each block has the master hash embedded in the header and we can verify each block in the mini-blockchain starting from the beginning, making sure the transactions in each block always correspond to the master hash in the previous block. Since there is a proof-of-work process required for each block before it will be accepted into the mini-blockchain, it becomes extremely difficult for an attacker to generate a fake mini-blockchain. Although difficult, it is far from impossible, and that's where the proof chain comes in.

We will talk about how the proof-of-work mechanism works in the next section when we look at the proof chain. It works a little differently compared to Bitcoin but it's the same basic principle; each block will take a certain amount of time to "solve" based on the current network difficulty and we can control the speed at which coins are created in that way. The difficulty is determined based on block timestamps like Bitcoin. Each time the mini-blockchain completes a cycle the difficulty should be recalculated.

The mining process should be essentially the same as Bitcoin but the issuance rate, issuance time period, and other such variables are up for discussion. I will talk a bit about some of these numbers near the end of this paper and which values I believe would work best. But we still have a problem here. If we build our copy of the account tree based on the integrity of the mini-blockchain, we must have a way to detect fraudulent mini-blockchain's in a highly secure manner, because new nodes have no history beyond the oldest block.

With Bitcoin we can start at the very beginning and work our way up to the latest point because we have the full blockchain. If the attacker creates a whole new mini-blockchain from the starting block, new nodes would have trouble telling it apart from the real mini-blockchain, because before that first block they have no history of what happened. The attacker can spend as much time as they need building up the cumulative difficulty of their mini-blockchain because it's not an ever-growing chain they have to outpace.

Even if difficult, an attacker could take as much time as he needed to form a fully valid mini-blockchain from the start, a chain with a higher cumulative difficulty than the real chain. He could then start broadcasting that mini-blockchain and it might propagate enough to impose a risk of becoming the main mini-blockchain. Thus we cannot rely on the cumulative difficulty of a mini-blockchain. We need some other mechanism which can perform the 2nd function of a blockchain and store a long term proof-of-work history.

The Proof Chain

The proof chain is a chain of proof-of-work solutions much like the blockchain is a chain of proofs, but in this case each proof is just two hashes and the entire chain could be stored for many decades before it grew more than a few gigabytes. However it is possible to trim the proof chain after a sufficient period of time. The proof chain, which provides our 2nd function of the blockchain, is truly finite. Each of our 3 mechanisms are light weight solutions and together they produce a very scalable and manageable system.

The way the proof chain works is that it essentially acts as a container of spent network energy because the proof-of-work solutions (or “proofs”) are chained together in such a way that each proof is connected to the one before it and each proof corresponds to the block it was created for. As noted each proof is made up of two hashes, the first hash in the proof is the hash of the last proof and the second hash is the hash of the block header. Unlike Bitcoin we don't just hash the block header looking for a solution.

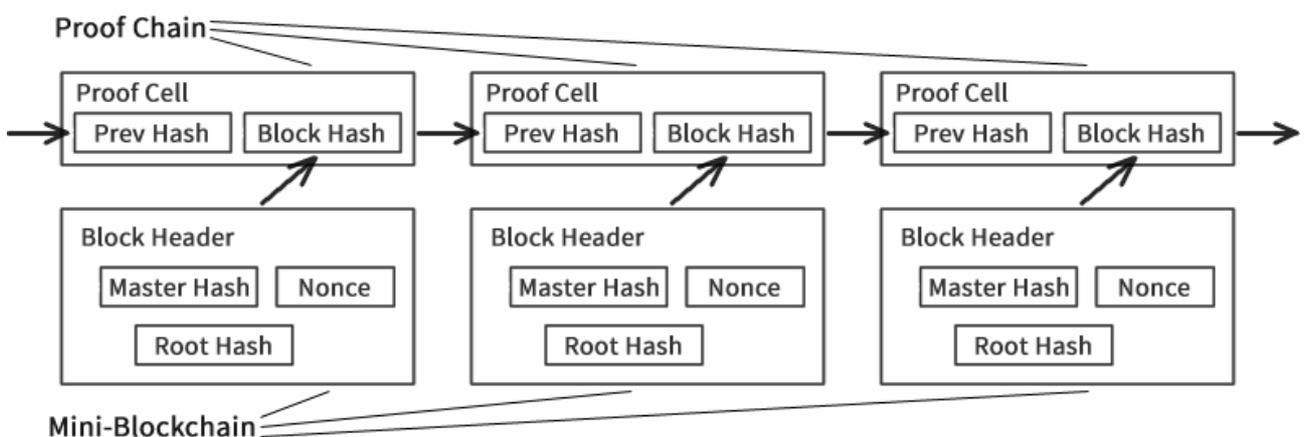


Figure 1-a

First we hash the block header and use that hash in our proof cell. If the proof results in a hash below the target the block is solved. If not, adjust nonce and try again. In this way it's possible to separate the proof-of-work into a self-contained chain. We are able to store this proof chain indefinitely because it is small and trimmable and we don't need to store all the

block data along with it. We can still have our mini-blockchain because proving the solution doesn't rely upon knowing the contents of the block. Each proof in the proof chain (the two hashes) feeds into the next proof, making it nearly impossible to generate a fake proof chain.

Of course this same thing could be achieved by having the block headers feed into each other just like Bitcoin, and saving the chain of headers instead of the "proof cells". However each proof cell should only contain two 256 bit hashes, making them even smaller than the block headers. We don't really need the master hash of the account tree to be in our proof cells because it can't really be used to prove anything beyond a certain time period. Plus the proof contains a hash of the block header and the header contains the master hash.

The master hash does need to be in the block headers as detailed in figure 1-a because it allows the nodes to verify the transactions in the block and make sure their account tree has been altered correctly by the block. The master hash stored in the block header is calculated after the transactions in the block have been applied to the account tree. The block headers don't need to feed into each other because the proof chain links them together. Before a miner can broadcast a new block they need a proof cell to go with it.

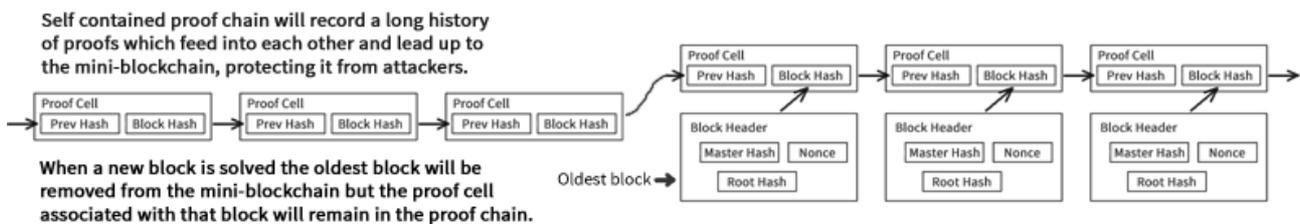


Figure 1-b

As figure 1-b demonstrates, we can pluck away old blocks from the mini-blockchain as it rolls along but we don't throw away the proofs associated with those blocks. Now we have some historical data about what happened before the oldest block in the mini-blockchain, and it's the best type of data: proof-of-work. We can work our way from the start of the proof chain up to where it feeds into the back-end of the mini-blockchain and we can verify that the oldest block in the mini-blockchain has the backing of the proof chain.

Now if an attacker tries to create a totally invalid mini-blockchain new nodes can see that there is little or no proof chain backing the fake mini-blockchain. The new node can also see that there is another mini-blockchain which has a very strong proof chain. The attacker does have to outpace the proof chain and it's highly unlikely that they would succeed at generating a proof chain with a higher cumulative difficulty than the real proof chain. The proof chain proves which mini-blockchain has the most long-term computational backing.

No longer does the attacker have forever to sit around generating a fake proof chain because the proof chain must feed into the mini-blockchain. If the attacker has no hope of generating a fake proof chain they could still attempt to build a fake mini-blockchain on the real proof chain. However that attack puts the attacker at an immediate disadvantage because they must start from the first block and catch up to the real mini-blockchain and then overtake it. But even if they some how manage that, they wont be able to keep it up.

Legitimate nodes which already have a copy of the real chain would see that the fake mini-blockchain is not real because it will contain invalid transactions which don't match their account tree. Soon after the attack stopped the legitimate nodes should once again cause the real mini-blockchain to overtake the fake mini-blockchain and things would go back to normal. The so called 51% attack would still be possible but the difficulty and consequences of such an attack should be about the same as Bitcoin.

Summary of Network Behavior

Network Synchronization

Network synchronization is achieved in 4 steps:

1. Acquire proof chain with the highest cumulative difficulty.
2. Acquire mini-blockchain which is associated with proof chain.
3. Acquire sector hashes associated with master hash in newest block.
4. Build account tree and verify each sector with sector hashes.

Now obviously it may be some what difficult to retrieve the sector data required to build the account tree from peers if the sector hashes are constantly changing each time a new block comes in. In order to retrieve the different sectors and complete step 4 the node will ask around for data which matches the sector hashes they acquired in step 3. No one is required to store old sector hashes but the node must build an account tree which is fully consistent with the master hash they have; they can't mix and match sectors associated with different master hashes.

However it's quite easy to solve this problem because in the time it takes to retrieve all the sectors from peers only a small fraction of the sectors would have been altered (assuming a non-adolescent system). The node can easily detect which sectors have changed by updating their list of sector hashes periodically. The node will always attempt to retrieve the newest sector data because most nodes are likely to have that data. The node will do this until the data in every sector matches its corresponding sector hashes and the latest master hash.

Notice this process does not rely upon much examination of blocks, one trusts the blocks they have because the proof chain backs them. It's extremely easy to verify the proof chain and once that is complete the node only needs to make sure the blocks it gets match the proof chain. As the account tree is being built the only thing which matters is that it ends up having the master hash of the latest block. Once that is complete and the node is synchronized it can begin to update the account tree normally by accepting valid blocks.

Transactions

Bitcoin keeps track of address balances simply by reading through the blockchain to see what has happened, it's a continual ledger instead of self contained address balances. Bitcoin transactions use a system which include "inputs" and "outputs" and the inputs of most new transactions usually reference the outputs of previous transactions. We can use the basic input and output concept for this scheme but the input will point to an account in the account tree and the output will also refer to an account in the account tree.

The input account will fund the transaction which is sent to the output account. This operation would cause the balance of the input account to decrease and the balance of the output account to increase. Fees would still be used as normal to give priority to transactions and provide incentive to miners. Obviously a transaction should not be accepted as valid if it reduces the balance of any account below 0, or if it requests anything which conflicts with the value of any balance, or it tries any other funny business.

In order to make sure the same signed transaction isn't processed by the network more than once, the signer of the transaction will need to include in the transaction the hash of any accounts referenced as inputs for the transaction (we don't need to worry about the outputs because we have no control over how those accounts change). After the transaction is applied the accounts will have a different hash and therefore nodes will know not to accept that same signed transaction again because they can see the account hashes don't match.

Thus attempting to use the same signed transaction a second time will not succeed because one cannot change the contents of a signed transaction. However the attacker would succeed if that account happened to have the same hash again at some later point in time, which would be considerably likely to happen unless we had something like the ref block field to ensure the hash of the account is nearly never the same. We could also just use an incrementing number instead of the ref block value but the ref block value provides some other useful information beyond being just a meaningless number.

Account Tree

Splitting the account data up into sectors may be tricky because as empty accounts are pruned from the account tree it will create "gaps" in the sectors which may fragment our tree and cause more sectors than we need. Shifting the position of all the accounts in the account tree in order to fill those empty gaps will cause the hash of all sectors to change and put unnecessary stress on network as a result. The obvious way to deal with this seems to be filling those gaps with new addresses which need to be added to the database.

That way we can fill the gaps and only alter the sector hashes of the sectors we fill in. If we manage to fill all the gaps in all sectors, or there are no gaps to fill then we can simply add any new accounts to the highest numbered sector. If a sector is filled a new sector is created. If a sector is totally emptied of all accounts it is removed, but only if it's the highest numbered sector, otherwise we just leave the sector alone and the gap filling process will fill the empty sector back up. The account tree is only altered each time a new block is accepted into the mini-blockchain.

There are no other times where the account tree should be altered in any way. The mini-blockchain acts to coordinate when the tree is updated. When a node receives a valid block they will carry out the transactions listed within the block by altering their account tree accordingly. After the alterations have been completed the node will recompute the sector hashes of any sectors which were changed in the process and then recompute the master hash and check it against the master hash listed in the block header.

We also need to store the number of the last block to have altered the account tree, which should be the same as the total number of blocks solved. We could include this value in the account tree or perhaps each block header could contain the block number. I believe Bitcoin doesn't need to do that because it has the whole blockchain and can simply count the number of blocks. Without knowing the total number of blocks we cannot calculate the total number of coins in circulation and cross-check it with balances in our account tree.

Proof Chain

An obvious question is how do we determine the difficulty of any given proof chain. If we were using a chain of block headers instead of a proof chain (which we could do) it may not be so tricky to assess the total cumulative difficulty of the chain. Since each proof in the proof chain represents a 256 bit number, one could look at the hash of the proof and get a fairly accurate reading on how much energy went into creating the proof. The proof itself should be the proof, not what is recorded in the proof.

To measure the cumulative difficulty of the proof chain we can break it up into segments (if it's too short we don't bother). The number of proof cells per segment would be the number of blocks between difficulty recalculation. Then we find the lowest difficulty proof in each segment to determine the "likely difficulty" of each segment. Finally, we sum up the likely difficulty values of each segment and that gives us a good approximation of the total cumulative difficulty of the proof chain in question.

The lowest difficulty proof within any given segment is going to have a value which closely corresponds to the actual target of that segment. That would make it extremely difficult for an attacker to create a fake proof chain with a misleading cumulative difficulty because each segment in the proof chain will only be as good as the weakest proof in that segment. Summing up the values derived from each segment will give us quite a high level of security because the longer the proof chain is, the higher the cumulative difficulty is.

It's possible to trim down the proof chain because after a sufficient period of time has passed it won't reduce the total cumulative difficulty of the chain enough to pose a problem. Assuming the trimmings are not done periodically the security risk should be low once the chain is long enough. There's no real reason to store proofs which go right back to original genesis block, we can simply update the genesis data in the source code after a sufficient period of time without requiring any type of forking to take place.

Discussing New Network Protocols

Dynamic Max Block Size

As noted in the introduction section of this paper there's a lot of heated debate concerning the maximum block size. I think one potential new network protocol worth discussing is the idea of a dynamic max block size. It could be made a floating value determined perhaps by several factors. Two methods which immediately come to my mind are 1) a mining-based voting system and 2) a system which measures how close the blocks are to their capacity. I believe a hybrid of these two systems would work best.

A mining-based voting system would allow the miners to place a vote value into their block headers. That vote value would be the maximum block size they want to see enforced. The max block size of the next block would be determined by averaging all the votes in the mini-blockchain and a block would not be accepted if it contains a vote which differs from that value by more than a certain percentage. The dynamic max block size would also need a lower limit to make sure mining pools don't push it down in an attempt to increase fees.

Such a voting system sounds feasible and it could allow us to manage the max block size through group consensus, but it gives groups such as mining pools a lot of power over what the max block size will be. Maybe if we used the voting system in conjunction with a system which measures how close the blocks are to their capacity we could counter any attempt to artificially lower the max block size. Perhaps the network could monitor the rate of zero fee transactions getting into blocks or something along those lines.

Even with our light weight scheme there does need to be a max block size because our network can only handle so much traffic before it virtually stops working. Even with a finite blockchain it can grow extremely quickly with large enough blocks. However in the future we may be capable of handling much larger blocks so I also think we need a way of changing it over time. As mentioned earlier, a hardfork is not very convenient. A voting system would be much more seamless and self-regulated.

Re-Mining "Lost" Coins

Although it is impossible to know when coins are truly lost compared to simply being held in long term savings, I don't really like the idea of knowing that the money supply will slowly get smaller and smaller until eventually there's only a few coins in circulation. Due to the divisibility of each coin it's unlikely we would ever lose enough coins to cause havoc on the system, because they would be lost over such a long period of time. However it's also unlikely we will ever find a way to completely stop losing our coins.

I personally don't like a money supply which increases forever or one which decreases forever. I think a stable money supply is the best option from an economic standpoint, assuming that money supply has sufficient granularity. I believe one possible way to achieve a stable money supply is to consider coins "lost" after a certain period of time has passed and the coins have not been touched. Once the coins are old enough they can be "re-mined" by miners using a special transaction which sends the funds to the miner.

Something like 100 years would be a good time period before coins are considered lost. We don't want to base our time calculation on the date when an address last received funds because such a system could easily be subverted by sending a small amount of coins to that address to "refresh" it. What we want to know is when funds were last transferred from an address to another address, and that's exactly what the ref block field tells us. The ref block field can be used to measure how long an account in our account tree has been inactive.

Deciding Technical Specifications

Account Tree

As previously noted, the accounts in the account tree are broken up into sectors and assigned sector hashes. I mentioned that each sector could hold something like 1000 accounts. I'm not an expert on hash trees but it's important we group together many accounts without linking together too many accounts. If we had something like 10,000 accounts in each sector, all our sector hashes would be changing far too quickly. However if we make each sector too small it becomes less efficient.

Of course we could increase the depth of the tree by grouping the sector hashes into sets of something like 100, producing a smaller set of hashes where each hash defines the state of an even larger portion of the account tree. Or we could go the other way and add more layers between the accounts and sector hashes. Before implementing this proposal, many features such as the exact structure of the account tree need to be fleshed out by people with experience in these areas so we get the most optimal solution.

Bitcoin already uses a Merkle tree structure for certain things so it's not something which should be outside the general experience level of a Bitcoin developer or people working on Bitcoin related technologies. There isn't anything in this proposal which requires the use of some entirely new technology, but the account tree is probably the biggest change this scheme has compared to most other crypto-currencies. Changing the transaction system to work with an account tree will be the hardest task.

Mini-Blockchain

If the proof chain is providing most of our security, at first glance it seems almost unnecessary to store anything beyond 1 or 2 blocks. It is of course necessary to have at least a few hundred because things such as the mining difficulty are determined by examining timestamp data from many blocks over a fairly long time period. In my opinion the mini-blockchain should contain at least 12 to 24 hours of history before trimming off old blocks. We also need to determine the max block size and at what speed blocks will be solved.

This light weight scheme gives us a lot of flexibility because we aren't bogged down by a full blockchain. The Bitcoin network tries to maintain an average of 1 block every 10 minutes where each block has a max block size of 1 MB. Assuming we don't go with a voting system for the max block size I would go with a max of at least 10 MB every 10 minutes. If our mini-blockchain is only saving blocks which record the last few dozen hours of history, 10 MB every 10 minutes is nothing.

I'm aware that having a small period of time between each block can have some side effects but the light weight design of this system may allow us to push this variable to the limits, but honestly I don't know. If we can manage to pull off 1 block per minute where each block has a max size of 1 MB we could make the mini-blockchain 1000 blocks long and it would hold 16.67 hours of history. If the difficulty is recalculated every time the mini-blockchain completes a cycle then it would change every 16.67 hours.

Coin Supply and Distribution

Bitcoin makes use of 2.1 quadrillion units where each coin is made up of 10 million units, resulting in a total of 21 million coins. Personally I think 1 quadrillion units is a much cleaner number. Each coin could be made up of 1 million units, yielding a total of 1 billion coins with 6 decimal places of precision. Nice clean numbers and very easy to remember. Each coin still has sufficient granularity with 1 million units, and 6 decimal places still provides enough precision without being excessively precise.

The time span over which bitcoins are released seems fine, I would prefer at least 100 years minimum before every last coin was mined. The distribution equation used by Bitcoin is not bad but when I look at the graph of how the coin supply increases over time I can't help but think perhaps the curve should be flattened out a little bit so that less of the coins are mined during the first quarter and especially near the very beginning when the difficulty is very low. Maybe we should be looking for more of a sigmoid shape in that graph.

I feel if we can achieve a fairer distribution near the beginning it will make late adopters feel better about it and result in a less speculative and volatile currency. It's not something I feel is absolutely necessary but if it can be done without too much extra work it should be done. It would also be preferable to use some sort of ASIC-resistant mining algorithm similar to Litecoin because that will also help to create a more level playing field, especially during the early stages. In any case, if this proposal works we will see which variants work best.

Conclusion

In this paper I have described a variant of the Bitcoin network protocol which is designed to eliminate the need for a full blockchain and significantly reduce the need for long term data storage. This is achieved by separating the functions of the blockchain into individual mechanisms optimized to perform certain tasks. Although still an untested concept, the result offers a purely P2P crypto-currency with many benefits such as increased block space, making room for extra functionality such as transaction messages/references.

The nature of the account tree and mini-blockchain also offer a greater level of privacy compared to Bitcoin because empty addresses are removed from the database and old blocks are discarded after a certain time period. We achieve a much greater level of scalability at the expense of possibly a slight drop in security, if any drop at all. The proof chain provides a high level of security for the system much like a full blockchain does, but at the same time it is super tiny and trimmable.

What will the future of crypto-currency look like? If this proposal is valid the game may have just changed forever. However it's one thing to make a proposal and another thing to follow through on that proposal and turn it into a reality. Satoshi is very respectable in that sense because he took the initiative to create a vastly complex system built on an array of new and exotic concepts which hadn't ever been tested before. If Satoshi really is one person it's a person with vastly inspiring dedication and spirit.

References

- [1] Nakamoto, S. 2008. Bitcoin: A peer-to-peer electronic cash system.
<http://bitcoin.org/bitcoin.pdf>
- [2] Blockchain.info. 2013. Blockchain Size Data.
<https://blockchain.info/charts/blocks-size>
- [3] Andresen, G. 2013. Bitcoin-Qt / bitcoind version 0.8.0 released.
<https://bitcointalk.org/index.php?topic=145184>
- [4] Reiner, A. 2012. Ultimate blockchain compression.
<https://bitcointalk.org/index.php?topic=88208>
- [5] Todd, P. 2013. Bitcoin Blocksize Problem Video.
<https://bitcointalk.org/index.php?topic=189792>
- [6] Bitcoin Wiki. 2013. Scalability.
<https://en.bitcoin.it/wiki/Scalability>
- [7] Rosenfeld, M. 2012. Are there any studies into the size of the blockchain scaling over time?
<http://bitcoin.stackexchange.com/questions/2798/>
- [8] Nakamoto, S. 2008. Re: Bitcoin P2P e-cash paper.
<http://www.mail-archive.com/cryptography@metzdowd.com/msg09964.html>
- [9] spartacusrex. 2013. Ideas / Possible Solution to Never-Ending Blockchain
<https://bitcointalk.org/index.php?topic=152219>
- [10] Bruce, J. 2013. Blockchain-less P2P Currency (theoretical idea)
<https://bitcointalk.org/index.php?topic=152662>
- [11] Bruce, J. 2013. Cryptocurrency with Finite "Mini-Blockchain"
<https://bitcointalk.org/index.php?topic=169311>