Subject: **Open sourced my Java SPV impl**

From: **Mike Hearn** <mike@plan99.net>
Date: Mon, Mar 7, 2011 at 2:13 PM
To: Satoshi Nakamoto <satoshin@gmx.com>

Hi Satoshi,

I hope you are doing well. I finally got all the lawyers happy enough to release BitCoinJ under the Google name using the Apache 2 license:

http://www.bitcoin.org/smf/index.php?topic=4236.0

It's incomplete - notably it doesn't properly handle block chain splits yet - but the rest is coming. I put a lot of work into documentation and comments so hopefully it'll open up BitCoin to a new audience who weren't able to understand/build the current code. Over the next month or two I'll be finishing off some of the bigger missing pieces for a full client-mode implementation.

I know you are busy right now but I'm hoping you can find time to answer a few questions I had.

As part of doing full SPV I'm thinking of adding a getmerklebranch message to the protocol. This would return a set of {blockhash, branch} pairs given tx hashes, so allowing verification of a broadcast transaction before it was incorporated into a block without storing the full chain. Does that approach sound good to you?

Also, I've been thinking of exploring different transaction types lately, eg by removing the IsStandard() checks for the testnet. It's clear you put a lot of thought into transactions beyond simply moving coins around up front, but unfortunately none of it was in the paper or documented in the code. Escrow, multi-pay and so on are all interesting but I was wondering if you could compile a list of ideas for things we can do with the scripting language at some point.

Finally, the code that allows for transaction replacement has been disabled but the comment doesn't say why. Is this just to reduce the attack surface/complexity or is there a deeper reason? I haven't fully understood why sequence numbers are a property of the tx inputs rather than the tx itself.

Hope you can find the time/energy to rejoin us soon! I don't know if you've seen this:

http://bitcoin.sipa.be/speed-lin.png

but it's exciting times for the network!

thanks!

/mike

From: **Satoshi Nakamoto** <satoshin@gmx.com>
Date: Wed, Mar 9, 2011 at 5:15 PM
To: Mike Hearn <mike@plan99.net>

I hope you are doing well. I finally got all the lawyers happy enough
to release BitCoinJ under the Google name using the Apache 2 license:

It's incomplete - notably it doesn't properly handle block chain
splits yet - but the rest is coming. I put a lot of work into
documentation and comments so hopefully it'll open up BitCoin to a new
audience who weren't able to understand/build the current code. Over
the next month or two I'll be finishing off some of the bigger missing
pieces for a full client-mode implementation.

That's great news! Much complexity can be left behind in a clean rewrite with only client requirements, and it opens it to Java developers too.

I know you are busy right now but I'm hoping you can find time to
answer a few questions I had.

I'm happy to answer any questions.

As part of doing full SPV I'm thinking of adding a getmerklebranch
message to the protocol. This would return a set of {blockhash,
branch} pairs

That's a CMerkleTx

given tx hashes, so allowing verification of a broadcast
transaction before it was incorporated into a block without storing
the full chain. Does that approach sound good to you?

I don't understand. A merkle branch links a tx back to a block, which only has significance if the block exhibits proof-of-work. Linking back to an as-yet unsolved block proves nothing.

Network nodes are able to verify 0-conf txes because they have the complete tx index, so they can:
1) verify signatures against dependencies.
2) say that they haven't seen another spend yet, because they know about every tx in existence.

Are you talking about CMerkleTxes for the tx's dependencies? That would get part 1), but not part 2).

If you don't know about all txes in existence, I don't know how to do 2). You could only rely on trusting other nodes for that. That trust can be distributed over multiple nodes. Nodes only relay transactions they accept as valid. If you receive inv messages for a tx from all the nodes you're connected to, they're attesting that it's valid and the first spend they saw.

Also, I've been thinking of exploring different transaction types
lately, eg by removing the IsStandard() checks for the testnet.

Very good idea. That should definitely be allowed on -testnet.

It's
clear you put a lot of thought into transactions beyond simply moving
coins around up front, but unfortunately none of it was in the paper
or documented in the code. Escrow, multi-pay and so on are all
interesting but I was wondering if you could compile a list of ideas
for things we can do with the scripting language at some point.

Finally, the code that allows for transaction replacement has been
disabled but the comment doesn't say why. Is this just to reduce the
attack surface/complexity or is there a deeper reason?

Just to reduce surface area. It wouldn't help with increasing tx fee. A tx starts being valid at nLockTime. It wouldn't work to have a tx that stops being valid at a certain time; once a tx ever becomes valid, it must stay valid permanently.

See these threads:
http://www.bitcoin.org/smf/index.php?topic=1786.msg22119#msg22119
http://www.bitcoin.org/smf/index.php?topic=2181.msg28729#msg28729

I haven't fully
understood why sequence numbers are a property of the tx inputs rather
than the tx itself.

It's for contracts. An unrecorded open transaction can keep being replaced until nLockTime. It may contain payments by multiple parties. Each input owner signs their input. For a new version to be written, each must sign a higher sequence number (see IsNewerThan). By signing, an input owner says "I agree to put my money in, if everyone puts their money in and the outputs are this." There are other options in SignatureHash such as SIGHASH_SINGLE which means "I agree, as long as this one output (i.e. mine) is what I want, I don't care what you do with the other outputs.". If that's written with a high nSequenceNumber, the party can bow out of the negotiation except for that one stipulation, or sign SIGHASH_NONE and bow out completely.

The parties could create a pre-agreed default option by creating a higher nSequenceNumber tx using OP_CHECKMULTISIG that requires a subset of parties to sign to complete the signature. The parties hold this tx in reserve and if need be, pass it around until it has enough signatures.

One use of nLockTime is high frequency trades between a set of parties. They can keep updating a tx by unanimous agreement. The party giving money would be the first to sign the next version. If one party stops agreeing to changes, then the last state will be recorded at nLockTime. If desired, a default transaction can be prepared after each version so n-1 parties can push an unresponsive party out. Intermediate transactions do not need to be broadcast. Only the final outcome gets recorded by the network. Just before nLockTime, the parties and a few witness nodes broadcast the highest sequence tx they saw.

From: **Mike Hearn** <mike@plan99.net>
Date: Wed, Mar 9, 2011 at 5:39 PM
To: Satoshi Nakamoto <satoshin@gmx.com>

If you don't know about all txes in existence, I don't know how to do 2). You could only rely on trusting other nodes for that. That trust can be distributed over multiple nodes. Nodes only relay transactions they accept as valid. If you receive inv messages for a tx from all the nodes you're connected to, they're attesting that it's valid and the first spend they saw.

Good point. I was talking about verifying the inputs yes, but it is indeed pointless unless you hear about all open transactions as well. So being able to fetch a CMerkleTx is not important.

Just to reduce surface area. It wouldn't help with increasing tx fee. A tx starts being valid at nLockTime. It wouldn't work to have a tx that stops being valid at a certain time; once a tx ever becomes valid, it must stay valid permanently.

See these threads:
http://www.bitcoin.org/smf/index.php?topic=1786.msg22119#msg22119
http://www.bitcoin.org/smf/index.php?topic=2181.msg28729#msg28729

I see. So right now fees are tricky because you have to decide up front what the fee should be, and if you guess too low, there's no way to correct the transaction and though the network will eventually forget it, your wallet still records that you spent the coins. This has already started happening.

It's for contracts.

Ah ha. A whole unexplored area of the system opens up before my eyes :-) The concept of forming distributed contracts and escrow transactions without needing to trust an intermediary is a concept nearly as novel as BitCoin itself, I think.

I have more questions!

There's an unfinished part of the protocol that deals with setting up publisher/subscriber channels for distributed routing via the network. What was the purpose of this? Was the idea to have a p2p market or did it have some kind of lower level function, like perhaps broadcasting expected tx fees?

There was an interesting discussion of generalizing BitCoin some months ago, but we struggled to fully understand how you planned to achieve it. I think I understood the concept of placing another merkle tree on top of multiple separate chains:

http://www.bitcoin.org/smf/index.php?topic=3414.msg48171#msg48171

But I didn't understand your comment about having 200 bytes for backwards compatibility. Also, I guess this is obvious, but to be super clear - in your idea the alternative chains would share exactly the same format and sets of verification rules as BitCoin (the same script language etc), so all miners can verify all blocks even if they are non-financial in nature? And then the point of having separate block chains is simply to manage storage costs and bandwidth for client-mode implementations?

Thanks!

From: **Satoshi Nakamoto** <satoshin@gmx.com>
Date: Wed, Mar 9, 2011 at 7:39 PM
To: Mike Hearn <mike@plan99.net>

> See these threads:
> http://www.bitcoin.org/smf/index.php?topic=1786.msg22119#msg22119
> http://www.bitcoin.org/smf/index.php?topic=2181.msg28729#msg28729

I see. So right now fees are tricky because you have to decide up front what the fee should be, and if you guess too low, there's no way to correct the transaction and though the network will eventually forget it, your wallet still records that you spent the coins. This has already started happening.

The network won't forget, and the owner's client will keep rebroadcasting it. The overflow transaction was remembered by the network for several months even as the remaining 0.3.8 nodes diminished.

Priority includes age, so as a transaction waits it ages and will eventually have enough priority.

See one of the links above where I contemplate sending an honest double-spend to increase the fee. It's a lot of work but could be done. I don't think it's worth it right now.

The current system, where nodes make sure to include enough fee for current conditions and the network makes sure all transactions get processed eventually, works well enough. Gavin is fixing the oversight where nodes didn't check the priority of their own transactions when writing them.

Users still worried about processing speed uncertainty should think of it as encouragement to include a fee.

There's an unfinished part of the protocol that deals with setting up publisher/subscriber channels for distributed routing via the network. What was the purpose of this? Was the idea to have a p2p market or did it have some kind of lower level function, like perhaps broadcasting expected tx fees?

I was trying to implement an eBay style marketplace built in to the client. Publish/subscribe would be used for broadcasting product offers and ratings/reviews. Your reviews would be weighted by the blocks you've generated. I rightly abandoned it in favour of JSON-RPC, so other authors could implement it externally. The publish/subscribe "meet in the middle" mechanism was an interesting concept, but nothing remains that uses it.

It was part of writing code to explore the most technically demanding use cases and make sure Bitcoin could support everything that might be needed in the future, given the locked-in nature of the rules once the block chain started.

There was an interesting discussion of generalizing BitCoin some months ago, but we struggled to fully understand how you planned to achieve it. I think I understood the concept of placing another merkle tree on top of multiple separate chains:

> http://www.bitcoin.org/smf/index.php?topic=3414.msg48171#msg48171

But I didn't understand your comment about having 200 bytes for backwards compatibility. Also, I guess this is obvious, but to be super clear - in your idea the alternative chains would share exactly the same format and sets of verification rules as BitCoin (the same script language etc), so all miners can verify all blocks even if they are non-financial in nature? And then the point of having separate block chains is simply to manage storage costs and bandwidth for client-mode implementations?

No, other chains do not follow Bitcoin's rules. They are completely independent chains. They share nothing except the miners. The other network's definition of proof-of-work is to make a solved (according to their own chain's difficulty) Bitcoin-format block that has a hash of their own block in it. They don't care if the Bitcoin block is valid or used by Bitcoin, but it allows miners to work both chains at once.

Procedure to hash a BitDNS block:
 - hash the BitDNS block
 - construct a Bitcoin block
 - insert the BitDNS hash into the scriptSig of tx 0 in the Bitcoin block
 - hash the Bitcoin block

The BitDNS block is valid if that hash is below BitDNS's target.

The BitDNS block needs to have with it about 200 bytes of data needed to reconstruct the Bitcoin block used in the hash:
 - the Bitcoin block header
 - the merkle branch to tx 0
 - tx 0 (btw, tx 0's prev hash is always 0 so leaving that out saves 32 bytes)

Note that it doesn't matter if the fodder "Bitcoin block" was actually valid in the Bitcoin chain, though it could have been. To BitDNS, it's just a bunch of salt necessary to do its convoluted hash calculation. If a miner is only mining for BitDNS and doesn't care about Bitcoin, it would use a blank Bitcoin block of all zeroes (except the nonce).

To further expand the idea for extensibility, consider instead of putting the BitDNS block hash in tx 0, you put the root of a merkle tree that includes BitDNS. This is the merkle tree that is conceptually at the top.

From: **Mike Hearn** <mike@plan99.net>
Date: Wed, Mar 9, 2011 at 7:52 PM
To: Satoshi Nakamoto <satoshin@gmx.com>

Thanks again.

Hal speculated that you intended to stash the new merkle root in the tx0 scriptSig. Good to know at least he had the right idea :-)