An Efficient Distributed Currency

Ben Laurie
(ben@links.org)

Sat Jul 23 15:48:59 2011 +0100 (22:b107be5bc87e)

1 Abstract

Given that it is probably impossible to create a decentralised currency[2], what's the next best thing?

I claim that it is a distributed currency: one that relies on a distributed central authority. In this case, the interesting questions are how one builds a distributed central authority, and how one chooses the participants.

I explore these questions and outline sketch solutions.

2 Required Reading

I will use the terminology and ideas already defined in [2]. I will not assume the existence of an efficient unbounded agreement protocol.

3 Building a Distributed Central Authority

We need an efficient way to agree the total state of the system (that is, what coins exist, who has possession of them and the transaction history[1]).

First, we view the state as a map of coins to purses. This can be represented as a list of coins, each with the number of the purse it is in. I call this a *snapshot*.

A snapshot can be hashed by forming an ordered list of the coins and building a Merkle tree from them. I call this a *snapshot hash*. Clients can now efficiently query the current state and check that the results match an agreed snapshot hash (I will come to how it is agreed later).

A transaction is a change in the state, which can be thought of as a transition from one snapshot to another. Only two transitions are legal.

First, creation of a new coin. This manifests itself by a new coin record appearing, assigning the coin to some purse.

Second, movement of a coin from one purse to another. This is simply a change to the appropriate coin record.

A transaction is recorded in the transaction log by appending the snapshot hash of the new snapshot (and remembering the corresponding state, so it can be queried!).

Leaving aside temporarily the question of how servers in the central authority (I shall call these *mintettes*) come to agree a transaction log, once this is in place we are now in quite a nice situation. At any point, a client can request from a mintette the current location of a coin and get a coin record, a snapshot hash and a proof that the coin record was included in the snapshot hash. If it was paranoid, it could also request the entire state for that hash and verify that the coin was not included twice. With this in hand, it could go to other mintettes and verify that the snapshot hash was current, or at least recent. Any mintette

[1] Although it makes sense to not keep a transaction history in some cases, my protocol requires it

that attempts to fib about the current agreed state will quickly be revealed by very cheap checks at other mintettes.

If mintettes also sign their responses, then any such naughtiness can be published as proof that the mintette should not be trusted. In a similar manner to the coin state, mintettes could keep a current list of blacklisted mintettes, including proofs of their malfeasance.

Furthermore, mintettes can verifiably enforce constraints on the system. For example, if we wanted to produce a new coin every ten minutes, as the Bitcoin[3] system does, mintettes would reject transactions that violated this requirement.

4 Agreement

We should not assume that absolutely every mintette is honest, but it seems reasonable to assume that a majority (or even a vast majority) are. This protocol is not a free-for-all. Mintettes will be part of a select, but hopefully disparate, group. Clients will be preconfigured with the list of mintettes (note that it is possible for the preconfigured list to show a more up-to-date list once the client is talking to them). If a mintette goes bad, it will hopefully be an isolated incident. But we must, nevertheless, detect and deal with it.

So, it is necessary that all honest mintettes agree the same transaction log. This can be achieved by using a Byzantine-fault-tolerant agreement protocol (for example, [1]) to agree an ordering over transactions they receive from clients. Obviously a server receiving a request to commit a transaction from a client should not acknowledge it until agreement has been reached with other mintettes. That agreement (perhaps combined with a state proof as above) is also sufficient to assure the client the transaction is valid, since mintettes should reject transactions that are not valid (at the point where it is agreed the transaction applies). Note that a client can safely inject multiple requests for the same transaction at different mintettes - the redundant ones will be rejected.

5 Coin Creation

I do not define a particular coin creation mechanism - anything the mintettes agree on is fine, of course, but as a proof of existence, here is one way to do it, which produces a similar outcome to the Bitcoin mechanism.

In order to create a new coin, first have all potential recipients register with the mintettes. The mintettes can agree a numbering for these recipients and choose one of them randomly to receive the new coin (using a commit-then-reveal protocol as described in [2]).

The mechanism for registration is left open, but could even, for example, be a proof-of-work: the mintettes publish a random block and in order to register clients must provide a unique hash collision based on that random block. This would be essentially equivalent to the Bitcoin mechanism, except that checkpoints would be a clear an explicit part of the system, and transactions would not need proof-of-work to incorporate them into the transaction log (which also

means transactions would complete more-or-less instantly instead of after a delay as they do in Bitcoin).

More efficiently, registration could be done in exchange for money, or proof of citizenship, or even simply the ability to maintain an open connection to a mintette[2].

6 Anonymity

I have left open the question of exactly how purses, coins and transactions are represented; there are many possibilities. One obvious option is that a coin is simply a serial number and a purse is a public key. In order to transfer the coin, a message containing the coin number and the new purse's public key, signed by the owning purse's private key would be sent to the mintette.

In order to provide anonymity[3] in this system all clients need do is create a new public/private keypair for each transaction in which they receive a coin: that is, each purse is only ever used once, to hold a single coin. This destroys any linkage between transactions and thus provides anonymity.

7 Conclusion

So now we can imagine a world in which mintettes are set up by a variety of organisations which have an interest in seeing a distributed currency succeed. They do not have to trust each other to operate the protocol, but, so long as a majority act honestly, the currency works correctly. Furthermore, unless the mintettes conspire, they do not have any say over what happens - that is in the hands of their clients, since each mintette must act strictly according to the rules, or risk being blacklisted by other mintettes.

8 Acknowledgements

Peter Eckersley, Chris Palmer and Adam Langley will probably recognise some of these ideas from a conversation we had in May 2011, on a related subject. I thank them for their insights, which I have shamelessly appropriated.

References

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Third Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, Louisiana, February 1999. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.

[2] Yes, this is also in effect a proof of possession of computing power and bandwidth: but at least it is only burnt during the issuance of coinage instead of forever!

[3] At least from the transaction log's POV: tracking may be possible in other ways, such as by observing IP addresses of clients.

[2] Ben Laurie. Decentralised currencies are probably impossible (but let's at least make them efficient). http://www.links.org/files/decentralised-currencies.pdf.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf.