ZeroTime: An approach to decentralized autonomous zero confirmation cryptographic-currency transactions.

Abstract

In this document I propose a system utilizing zero confirmation off-chain transactions without the possibility of a double-spend. While we appreciate the InstantX[3] technology it suffers several major weaknesses in that it is a centralized 3rd party approach vulnerable to collusion attacks, uses a 1st generation approach to P2P network routing that yields both a limited horizon and overall limited scalability. In order to achieve a tamper-less short-term consensus the decision making must be left to autonomous voters that are made up of non-deterministically selected non-biased nodes that maintain virtual synchrony[1] in a fast and decentralized manner.

Definitions

- Client - A network node that does not share, route or relay data.
- Peer - An unreachable network node that shares, routes and relays data.
- Super Peer - A reachable network node that shares, routes and relays data.

Background

Peercoin uses Proof-of-Work and Proof-of-Stake to maintain consensus throughout it's peer network. Due to hard-coded variables the time for a

transaction to be considered confirmed and safe guarded from a double-spend attack is fixed.

We propose a solution to this through the use of a global transaction pool concurrency mechanism to lock transaction inputs across the entire network via a hybrid TCP/UDP protocol that forms consensus and synchrony on global transaction pool states.

By using a virtual synchrony model, *it is relatively easy to maintain fault-tolerant replicated data in a consistent state. For example, tools for building distributed key-value stores, replicating external files or databases, locking or otherwise coordinating the actions of group members, etc.*

Virtual synchrony replication is used mostly when applications are replicating information that evolves extremely rapidly. The kinds of applications that would need this model include multiuser role-playing games, air traffic control systems, stock exchanges, and telecommunication switches. Most of today's online multiuser role-playing games give users a sense that they are sharing replicated data, but in fact the data lives in a server on a data center, and any information passes through the data centers. Those games probably wouldn't use models like virtual synchrony, at present. However, as they push towards higher and higher data rates, taking the server out of the critical performance path becomes important, and with this step, models such as virtual synchrony are potentially valuable.[1][4].

Necessary Conditions for Successful Double-Spending[2].

To perform a successful double-spending attack, the attacker A needs to trick the vendor V into accepting a transaction TRV that V will not be able to redeem subsequently. While this might be computationally challenging for A to achieve if TRV was confirmed in a Bitcoin block7, this task might be easier if the vendor accepts fast payments.

In this case, A creates another transaction TRA that has the same inputs as TRV (i.e., TRA and TRV use the same BTCs) but replaces the recipient address of TRV—the address of V— with a recipient address that is under the control of A.

Note that our analysis is not restricted to the case where the recipient address is controlled by A and applies to other scenarios, where the recipient is another merchant.

In this respect, let tVi and tAi denote the times at which node i receives TRV and TRA, respectively. As such, tV and tAV denote the respective times at which V receives TRV and TRA. The necessary conditions for A's success in mount- ing a double-spending attack are the following:

Requirement 1 — tV < tAV : This requirement is essential for the attack to succeed. In fact, if tV > tAV , then V will first add TRA to its memory pool and will reject TRV as it arrives later. After waiting for few seconds without having received TRV , V can ask A to re-issue a new payment.

Our solution to the zero confirmation double-spend problem is straightforward, obtain a global lock on the transaction, evict consensus conflicts from the global transaction pool and reject blocks that contain these conflicting transactions.

Sender Procedures

1. Broadcast.

   Broadcast the transaction to the entire network with the intent of zero confirmations. This will propagate the message both over TCP and UDP reaching a majority (50.01%) of the network backbone typically in about one second. The global transaction pool will lock the transaction inputs and nodes will start forming a consensus vote resulting in the eviction of conflicting transactions.

2. Probe.

   Once a transaction has reached at least a majority of the network the node MUST pick one storage node from each slot in the system and probe each of their transaction pools for a transaction lock matching the transaction id in question and await a response.

3. Completion.

   Once at least majority of the storage nodes have responded confirming their lock on the transaction it is considered confirmed and safely spendable by the recipient.

Recipient Procedures

1. Probe.

   Upon reception of the transaction and once it has reached at least a majority of the network the node MUST pick one storage node from each slot in the system and probe each of their transaction pools for a transaction lock matching the transaction id in question and await a response.

2. Completion.

   Once at least majority of the storage nodes have responded confirming their lock on the transaction it is considered confirmed and safely spendable.

Network Overview

The core protocol supports three tiers made up of clients, peers and super peers. This allows for maximum scalability while not relying on any centralized network components and permitting full autonomous operation.

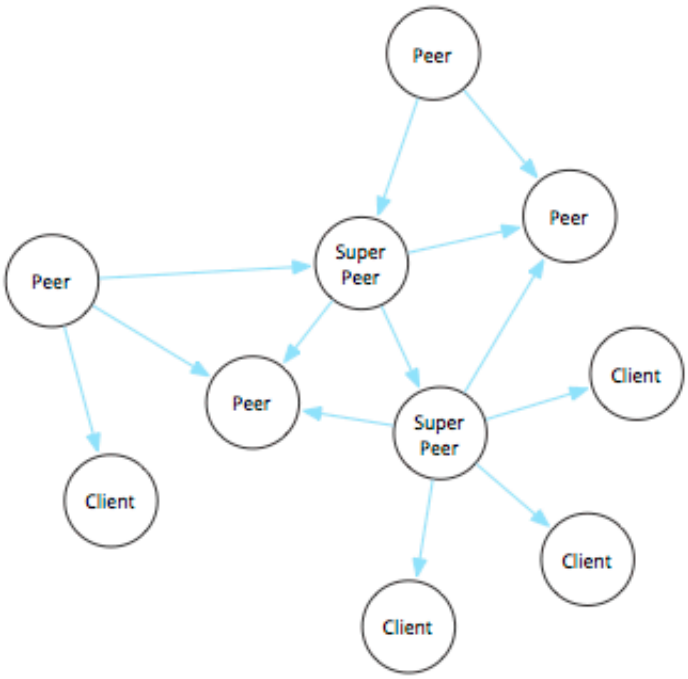


Figure 1. An autonomous 3-tier P2P network.

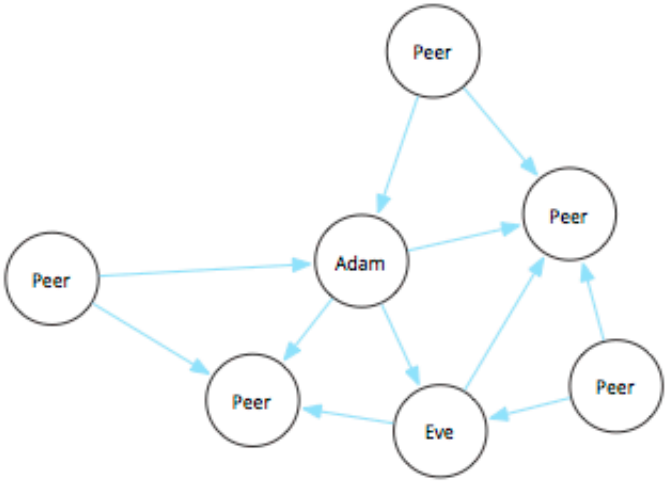
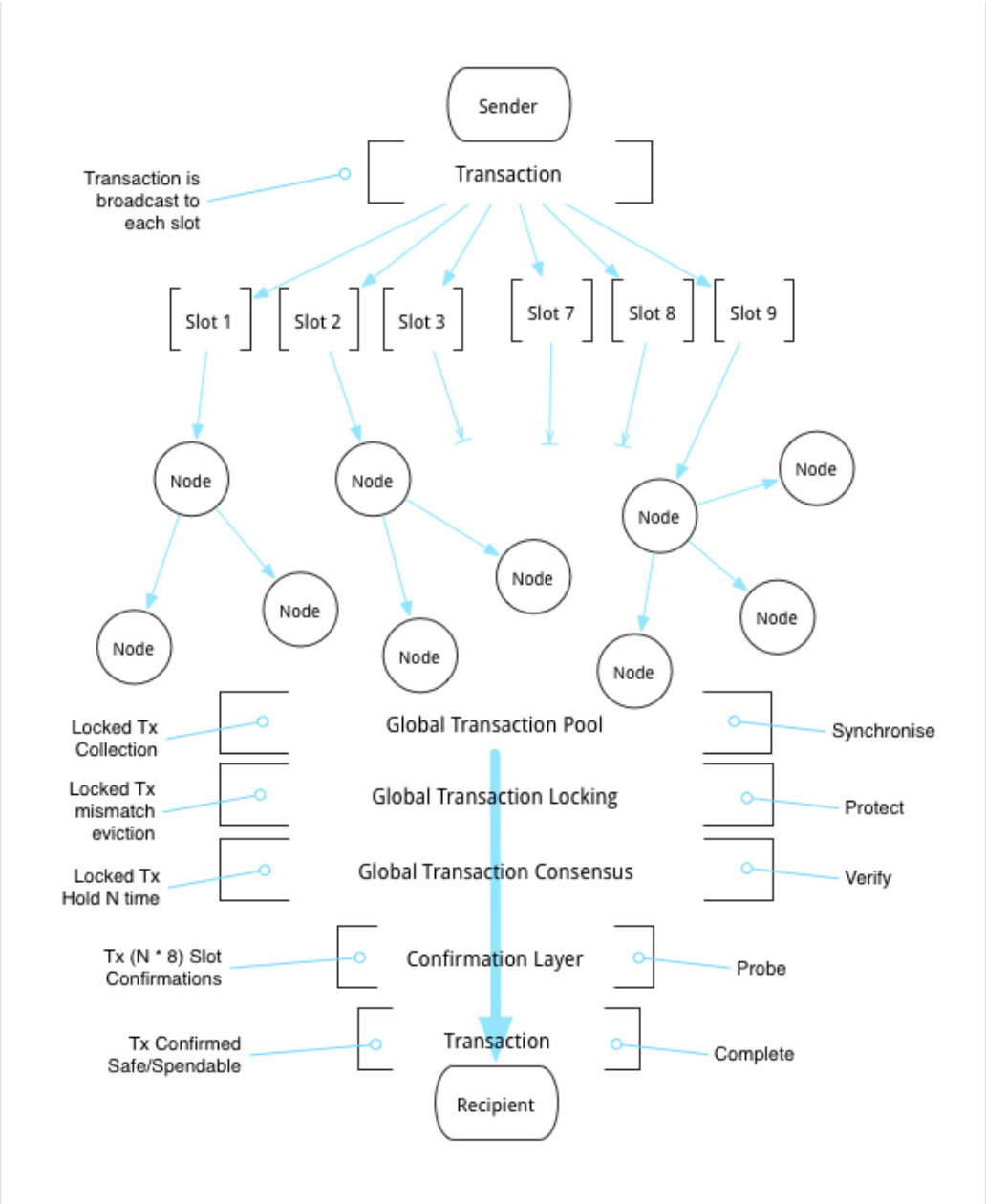


Figure 2. A 3rd party controlled P2P network.

A zero confirmation transaction requires much more complexity than a single

confirmation transaction by using only slightly more network load.



Security Considerations

The inability to forge identities in the UDP routing layer prevents Sybil attacks and the structure of the network was designed to deal with common attacks found in

DHT-like systems.

Conclusion

With our proposal we have satisfied the requirements which are essential for preventing a double-spend attack via a global transaction pool consensus and locking mechanism. Additionally, a UDP routing layer was introduced which allows for the fastest possible data paths allowing for maximum scalability. Simulation results show the average real-world time for zero confirmation transaction completion is around 8 seconds.

Author

John Connor

Public Key:

```
047d3cdc290f94d80ae88fe7457f80090622d064757
9e487a9ad97f77d1c3b3a9e8b596796ebeb23a78214
fc0a95b6a093b3f1d5e2205bd32168ac003f50f4f557
```

Contact:

```
BM-NC49AxAjcqVcF5jNPu85Rb8MJ2d9JqZt
```

References

1. https://en.wikipedia.org/wiki/Virtual_synchrony
2. https://eprint.iacr.org/2012/248.pdf
3. https://www.scribd.com/fullscreen/241012134
4. https://en.wikipedia.org/wiki/QuickSilver_(project)