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 pseudodeterministically[5][6] 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.
- Lock - An object that protects a particular transaction's input(s) from being re-spent by the sender prior to block inclusion.
- Vote - A vote in favor of a particular lock.
- Question - A question sent to peers regarding state of a particular transaction's input(s).

- Answer - An answer received from peers regarding the state of a particular transaction's input(s).
- Score - A 16-bit value describing the relevance of a peer's vote.
- K - A variable defining the number of closest votes to honor.

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. Additionally a recipient interrogation protocol is devised.

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 mounting 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's input(s), evict consensus conflicts from the global transaction pool and reject blocks that contain these conflicting transactions. Lastly the recipient performs interrogatories to confirm the transaction's input(s) are protected from being spent again before the next block event occurs.

General Overview

In order to send a ZeroTime protected transaction you would first broadcast it to the network, after a short interval you would broadcast a lock. Finally peers start forming consensus votes if their current score is at least zero. Once the K closest

votes to the current block height are received the global transaction pool will lock the transaction's input(s) evicting any conflicting locks that may exist. At this point the input(s) cannot be spent under any circumstances. If an attacker arbitrarily inserts locked inputs into a block it will be rejected via consensus conflict. If a malicious peer attempts to create a longer chain of work, the main chain while possibly shorter in proof still has the greater trust because of the consensus state held in each peers memory outweighs that of the attacker. Because of this malicious block reorganizations are not possible. Because this consensus state remains for some time the malicious peer SHOULD be banned from the network. The final step is that the recipient forms connections to N random recent peers that it is not currently connected to confirm the state of the locked input(s). The attacker as no ability to control this final step and if a lock conflict is discovered the transaction MUST wait to be consolidated during the next block event. The system is designed to force malicious peers to isolate themselves from the consensus making attacks extremely difficult to setup, costly to carry out and impossible to sustain. Should an attacker succeed in circumventing all intermediate steps the final step will result in the transaction remaining unconfirmed indefinitely from the recipients view.

Network Overview

The core protocol supports three tiers made up of clients, peers and super peers. These nodes communicate over both TCP and UDP. The UDP routing layer allows for one-hop lookups and wide-area message broadcasting. This allows for maximum scalability while not relying on any centralized network components and permitting full autonomous operation while allowing for the fastest network propagation times possible.

Figure 1. An autonomous 3-tier P2P network.

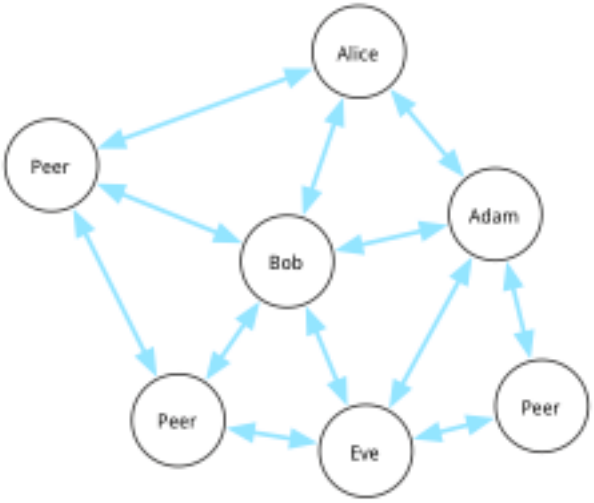
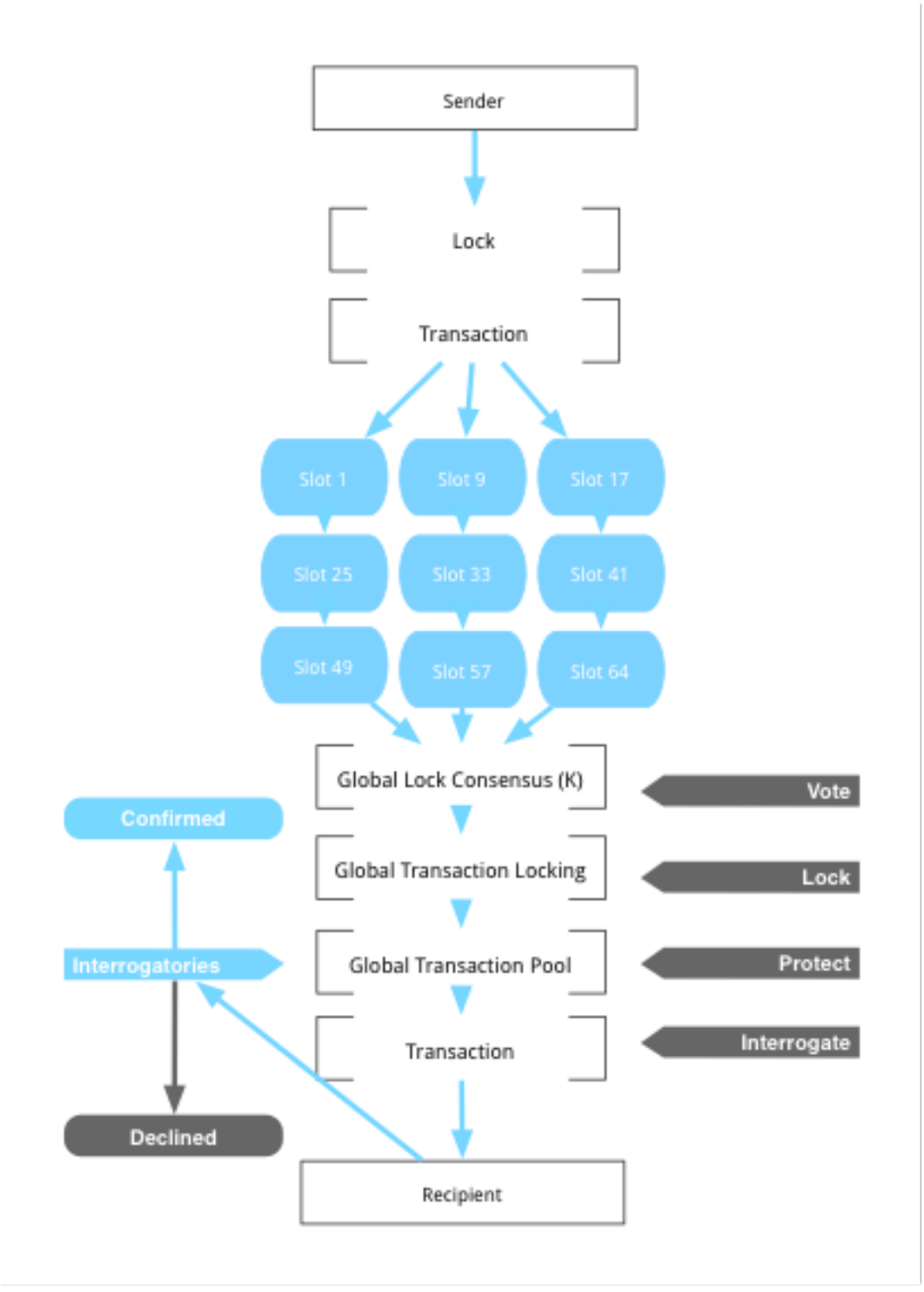


Figure 2. A 3rd party controlled P2P network.

A zero confirmation transaction requires much more complexity than a single

confirmation transaction by using only slightly more network load. The time to confirmation is left up to the recipient, however voting, locking and eviction take on average 372 milliseconds.


Sender
Lock
Transaction
Slot 1
Slot 9
Slot 17
Slot 25
Slot 33
Slot 41
Slot 49
Slot 57
Slot 64
Global Lock Consensus (K)
Vote
Confirmed
Global Transaction Locking
Lock
Interrogatories
Global Transaction Pool
Protect
Transaction
Interrogate
Declined
Recipient

Operations

1. Vote Score Calculation.

   To calculate the score for a given vote the public key used to sign the vote is hashed. The resulting hash is hashed with the vote block hash and the 16-bit representation of the result is calculated by subtracting the lesser of the two from the greater.

   Example:

   ```
   int16_t score = -1;

   uint32_t index = block_index_by_height(
       ztvote.block_height()
   );

   if (index->block_hash() == ztvote.hash_block())
   {
       uint8_t * digest1 = sha256d(ztvote.public_key().hash())

       uint8_t * digest2 = sha256d(index->block_hash());

       sha256 hash2 = sha256_from_digest(&digest2[0]);

       uint8_t * digest3 = sha256d(&digest2[0], &digest1[0]);

       sha256 hash3 = sha256_from_digest(&digest3[0]);

       score =
           (hash3 > hash2) ?
           (hash3 - hash2).to_int16() : (hash2 - hash3).to_int
       ;
   }
   ```

2. Determine K Closest Votes

   To determine the K closest votes to the the current block a

pseudodeterministic[5][6] Bellagio Algorithm is used. The XOR metric of the block height and score is calculated. Next the list is sorted and the top K entries are retained.

Example:

```
array<uint32_t> closest;

array<int16_t> vote_scores;

uint32_t block_height = best_block_height();

map<uint32_t, uint32_t> entries;

for (uint32_t & i : vote_scores)
{
    uint32_t distance = block_height ^ i;

    entries.insert(pair(distance, i));
}

for (pair & i : entries)
{
    closest.insert(i.second);

    if (closest.size() == k)
    {
        break;
    }
}
```

Sender Procedures

1. Broadcast Transaction.

   Broadcast the transaction to the network.

2. Broadcast Lock.

Once the transaction has been broadcast either the sender or the recipient MUST broadcast a lock on the transaction's input(s).

3. Perform Vote.

   When a peer receives a lock and it's calculated score is at least zero it SHOULD form a vote on the lock. Votes with negative scores will be dropped by remote peers. Once the desired percentage of the K closest votes to the current block height are tallied conflicts are resolved and the transaction's input(s) are considered locked.

Recipient Procedures

1. Broadcast Lock.

   The recipient MAY OPTIONALLY broadcast a lock on the transaction's input(s).

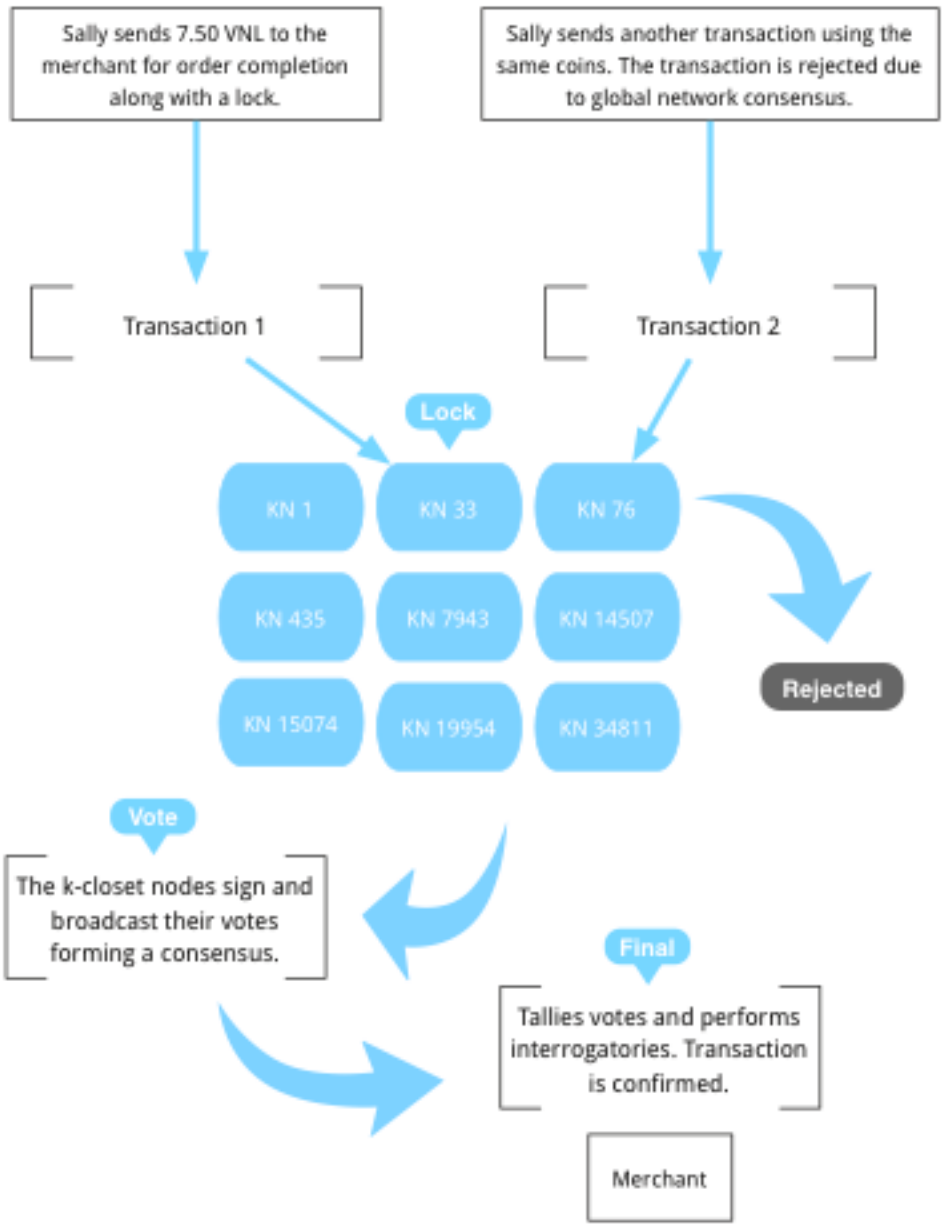
2. Perform Vote.

   When a peer receives a lock and it's calculated score is at least zero it SHOULD form a vote on the lock. Votes with negative scores will be dropped by remote peers. Once the desired percentage of the K closest votes to the current block height are tallied conflicts are resolved and the transaction's input(s) are considered locked. Lastly, the transaction's input(s) MUST be interrogated.

3. Perform Interrogatories

   The recipient MUST send a question to N peers that it is not currently connected to. Once some number of correct answers are received the transaction's input(s) are considered confirmed and safely spendable. The variable N MUST be able to be adjusted to allow for greater speed or longer wait times depending on the use case of the recipient.

Example Transaction



1. Sally sends 7.50 VNL to merchant for order completion.
2. Sally locks the transaction's input(s).
3. Nodes form votes, resolve conflicts and lock the input(s).
4. Sally sends the same coins to herself and it is rejected.

5. The merchant performs interrogatories and the transaction is confirmed.
6. Sally receives her order from the merchant.

Network Attacks

1. Vote Rigging

   In order for an attacker to rig the voting system they must control at least 50% of the K closest votes to the current block height of the recipient of a transaction. We can calculate this as (((1 in N where N = Total Network Nodes) / K) / 32767.5). Therefore the probability of an attacker owning 50% of the transaction recipients K-closest votes is approximately (((1 in 350) / 64) / 32767.5) / (score > -1) ? inf : 1 = 1.36e-09 on a 350 node system where K = 64 and the score is greater than zero. Regardless of these odds if a lock conflict occurs they will cancel each other out.

2. Lock Race Attack

   In a lock race attack the sender forms two transactions and two locks and submits one to the merchant and the other to the rest of the network. In this case the locks will cancel each other out and the transaction MUST wait for block inclusion before being considered confirmed.

3. Finney Attack

   In a Finney Attack an attacker inserts an arbitrary transaction into a block sending funds to themselves but instead of broadcasting the block they spend these inputs at a merchant. Before the merchant's transaction is included into a block the attacker broadcasts his block reversing the payment to the merchant.

   In order to stop these attacks blocks that contain conflicting transaction inputs MUST be rejected by the locking system.

4. Vector76 attack

In order for this attack to succeed a Finney Attack must succeed.

5. Network Split Attack

   In order to split the consensus of the network a Lock Race Attack must succeed. Because conflicting locks cancel each other out this attack becomes ineffective.

Security Considerations

None

Conclusion

With our proposal we have satisfied the requirements which are essential for preventing a double-spend attack via a global transaction pool consensus, locking and interrogation 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 one second but ultimately left up to the recipient's interrogation procedures.

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)
5. http://eccc.hpi-web.de/report/2012/101/download/
6. http://drops.dagstuhl.de/opus/volltexte/2012/3443/pdf/59.pdf