Node Incentives: An autonomous decentralised incentive and donation system.

Abstract

In this document I propose a autonomous decentralised incentive and donation system using the security of the blockchain to determine the winner.

Definitions

- Client - A network node that does not share, route or relay data.
- Peer - An unreachable network node that shares, routes and relays data.
- Public Peer - A reachable network node that shares, routes and relays data.
- Vote - A vote in favor of a particular address and block height.
- Question - A question sent to peers regarding it's incentive status.
- Answer - An answer received from peers regarding it's incentive status.
- 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

Historically the more incentive a user has to participate in a peer-to-peer network the higher the chances they will. It could be free phones calls, movies or anonymizing services, etc that are provided in exchange for supporting the network infrastructure. In the case of cryptographic currencies naturally it should be a portion of the coinbase reward. As mining pools have centralized the reward is given to the least useful nodes on the network. While pool nodes do consolidate transactions they typically have limited resources or are firewalled and unreachable.

General Overview

Spreading part of the coinbase reward to a series of winning nodes that are performing network services (hosting mobile clients or sharing the blockchain) to incentivize participants to allow their nodes to act as public peers will resolve scalability issues due to lack of publicly accessible nodes. In order to incentivize participants in supporting the network infrastructure by acting as a public peer a portion of the coinbase reward MUST be donated to a subset of vote winners after the solving of Proof-of-Work blocks.

But there are no similar incentives for individuals or businesses running full nodes. They have only the gratification that they are supporting the network. And if they are not mining, others are being rewarded for it. This is a problem, as nodes are arguably as important as mining. Keeping a full node running for an extensive period entails cost. The hardware being used may be out of play for anything else, or the cost of electricity is significant after an extended time.[1]

In order to participate in the system it MAY be required for the node to maintain a wallet balance that meets or exceeds a globally defined minimum amount. Client nodes do not participate in the system and therefore are not required to implement the algorithms, instead they follow the longest chain via their connected public peers.

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(
    vote.block_height()
);

if (index->block_hash() == vote.hash_block())
{
    uint8_t * digest1 = sha256d(vote.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. Node Score Calculation.

   To calculate the score for a given node the endpoint is hashed. The resulting hash is hashed with the current block's 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 = best_block_height();

uint8_t * digest1 = sha256d(node.endpoint().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_int16()
;
```

3. Determine K Closest Recent Nodes

   To determine the K closest recent nodes to the the current block a pseudodeterministic[2][3] 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> node_scores;

uint32_t block_height = best_block_height();

map<uint32_t, uint32_t> entries;

for (uint32_t & i : node_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;
    }
}
```

4. Voting

   When a new block is detected pick the K (where K = 2) closest recent nodes to the (current block height + 2). If the (current block height + 2) is odd the first closest node MUST be used likewise if the (current block height + 2) is even the second closest node MUST be used. If your vote score is at least zero you SHOULD broadcast a vote for the (current block height + 2) and the address of the donation recipient previously obtained from an answer. If your vote score for block (current block height + 2) is less than zero your votes will be rejected by the network consensus.

   Example:

```
array<node> closest;

uint32_t block_height = best_block_height() + 2;

array<node> nodes;

map<uint32_t, uint32_t> entries;

for (uint32_t & i : nodes)
{
    uint32_t distance = block_height ^ i.score();

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

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

    if (closest.size() == 2)
    {
        break;
    }
}

pair<block_height, node> winner;

winner.first = block_height;

if ((block_height & 1) == 0)
{
   winner.second = closest[0];
}
else
{
   winner.second = closest[1];
};
```

5. Tally

To tally votes take the address and block height with the most votes.

Example:

```
map<uint32_t, map<address, array<incentive_vote> > > votes;

map<address, array<incentive_vote> > block_votes =
    votes[best_block_height()]
;

address winner;

uint32_t most_votes = 0;

for (vote & : block_votes)
{
    if (vote.second.size() > most_votes)
    {
        most_votes = vote.second.size();

        winner = vote.first;
    }
}
```

6. Transaction

   The donation is paid out by the Proof-of-Work miners and enforced by global network consensus. If a miner fails to include an incentive transaction the block MUST be rejected.

   RPC extensions to getblocktemplate:

   1. incentive.enforced - If true incentives are enforced via global network consensus.
   2. incentive.address - The address to receive a part of the coinbase reward.
   3. incentive.amount - The amount of the coinbase to be awarded to the address.

Example:

```
    "result": {
        "version": 4,
        "previousblockhash": "000000000000ec08f...ce96c8bbf2",
        "transactions": [],
        "coinbaseaux": {
            "flags": "062f503253482f"
        },
        "coinbasevalue": 74074075,
        "target": "000000000002134c00000...00000000",
        "mintime": 1441262736,
        "mutable": [
            "time",
            "transactions",
            "prevblock"
        ],
        "noncerange": "00000000ffffffff",
        "sigoplimit": 60000,
        "sizelimit": 3000000,
        "curtime": 1441263729,
        "bits": "1b02134c",
        "height": 197546,
        "incentive": {
            "enforced": true,
            "address": "VdfoLns3UFSqd5QFhZecBvoNV9Afs3EgdG",
            "amount": 3703703,
        }
    },
    "id": "0"
```

In the example above 5% of the coinbase reward is donated to the block winner. The exact percentage is implementation dependent but up to 50% of the coinbase reward is acceptable in terms of cost/labor.

Procedures

1. Vote.

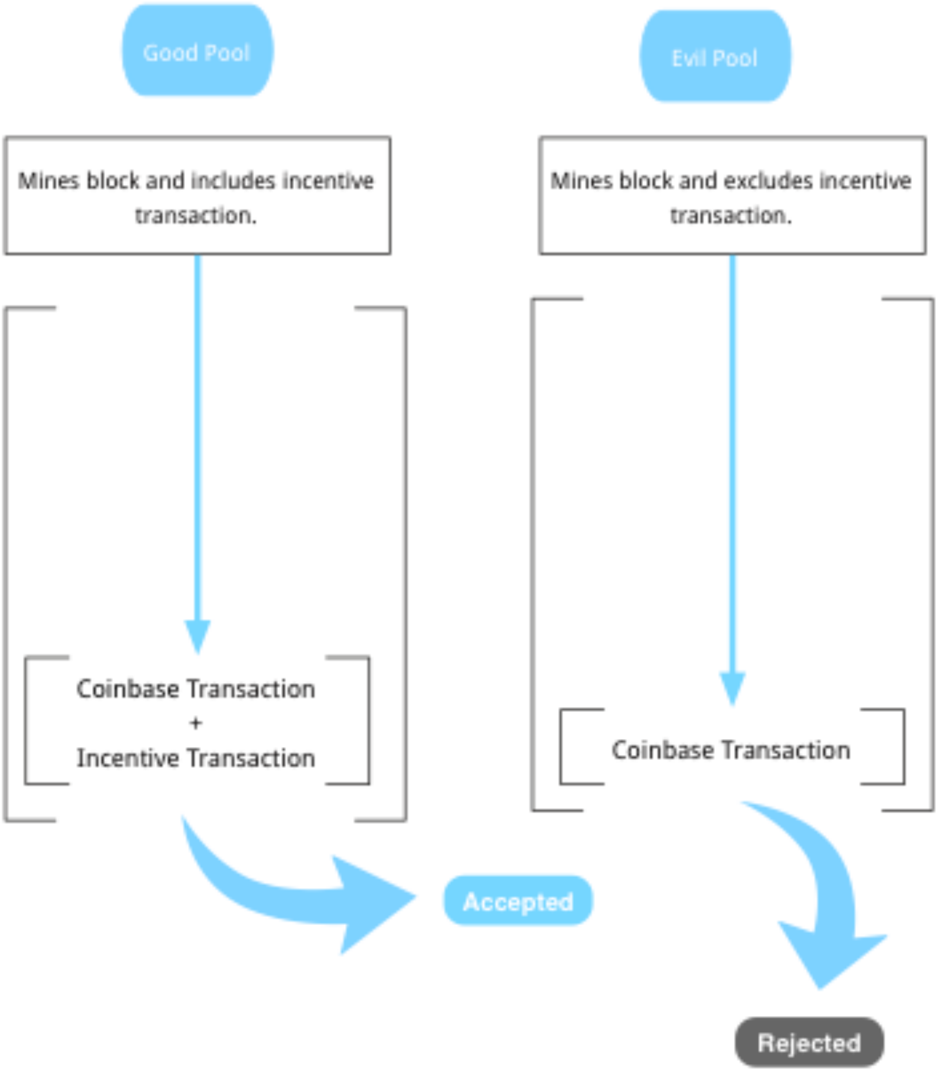
When a new block is detected you SHOULD perform a vote.

2. Tally

   Once enough votes have been received for the next block they may be tallied.

3. Enforce.

   Reject blocks that do not include the incentive transaction to the awarded address.


Good Pool
Evil Pool
Mines block and includes incentive transaction.
Mines block and excludes incentive transaction.
Coinbase Transaction + Incentive Transaction
Coinbase Transaction
Accepted
Rejected

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 + 2. 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 a current block height + 2's K-closest votes is approximately (((1 in 350) / 2) / 32767.5) / (score > -1) ? inf : 1 = 4.35e-08 on a 350 node system where K = 2 and the score is greater than zero.

If wallet balance enforcement is utilized the more it cost financially to attack the system.

Security Considerations

None

Conclusion

With our proposal we have satisfied the requirements which are essential for building an autonomous decentralised incentive and donation system. Additionally, we have discovered that the system can be used for making other types of reward programs in autonomous decentralised network environments.

Author

John Connor

Public Key:

```
047d3cdc290f94d80ae88fe7457f80090622d064757
9e487a9ad97f77d1c3b3a9e8b596796ebeb23a78214
fc0a95b6a093b3f1d5e2205bd32168ac003f50f4f557
```

Contact:

```
BM-NC49AxAjcqVcF5jNPu85Rb8MJ2d9JqZt
```

References

1. https://bitcoinmagazine.com/19620/bitnodes-project-issues-first-incentives-node-operators/
2. http://eccc.hpi-web.de/report/2012/101/download/
3. http://drops.dagstuhl.de/opus/volltexte/2012/3443/pdf/59.pdf