Vanillacoin: An efficient decentralized currency for the internet.

Abstract

In this document I propose an energy efficient cryptographic internet currency with a strong network security model, sub-second transaction times and low long-term inflation.

Background

All currently existing cryptographic currencies derived from and including Bitcoin[4] have inefficient network implementations that send all packets as plain text. All currently existing cryptographic currencies derived from and including Peercoin[5] implement Proof-of-Stake algorithms that are not energy efficient, utilizing excess CPU cycles while staking. Vanillacoin was built to solve these problems while bringing a modern day approach to it's design.

Cryptographic Implementation

The core cryptographic algorithms are similar to that of Bitcoin[4] and Peercoin[5]; therefore the they remain well tested and secure.

Network Implementation

The core Bitcoin[4] and Peercoin[5] networking code is an example of poor and outdated design. The Bitcoin wiki[6] points out only some of the problems:

Blocks are taking over a second, on average, to process once downloaded. Also,

testing reveals very large queues of blocks being processed per message loop, which is not what you would expect if the thread was pulling them out of the queue as they arrive on the sockets.

It is also points out that:

...message queues are processed to completion, one at a time per node. This can result in big backups of messages from other nodes.

Also, packets are sent as plain text and peers listen on pre-defined port numbers. At one time Satoshi Nakamoto thought about resolving some of these network concerns in the Bitcoin[4] protocol.

On december 9th 2009 Satoshi Nakamoto said:

I have thought about eventually SSLing all the connections. I assume anything short of SSL would be pointless against DPI.

On december 9th 2009 Satoshi Nakamoto said:

...the other stealth stuff would be kinda pointless if it's always the same port number.

We avoid these problems altogether with a modern approach and make the network as "real-time" as possible. Some of the following steps were taken:

1. Single Threaded or Multi-Threaded

   Because of the asynchronous design a single network thread is able to operate more efficiently than multiple network threads on the typical computer, tablet or phone while using a "thread per core" design is better suited for servers such as exchanges that need to process 1000's of transactions per second. Therefore we have provided both modes of operation.

2. Asynchronous

   All network calls occur asynchronously by being handed off to the underlying

operating system.

3. Encryption

   All TCP network connections are secured using the Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)[1]. All UDP communications are encrypted using the Stream Cipher HC-256[3]. This prevents data mining, eavesdropping, censorship, traffic shaping and Deep Packet Inspection(DPI)[7].

4. Random ports

   Random ports are used as to not advertise to the world that we are operating a cryptographic currency node.

5. UDP Layer

   Broadcasting messages over UDP enables them to reach more peers quicker using less bandwidth while O(1) routing allows for maximum scalability as the network grows.

   Example:

```
/**
 * The number of slots per block.
 */
enum { slots_per_block = 8 };

/**
 * The number of blocks in the system.
 */
enum { total_blocks = 8 };

/**
 * The number of peers in a slot.
 */
enum { peers_per_slot = 64 }:

/**
```

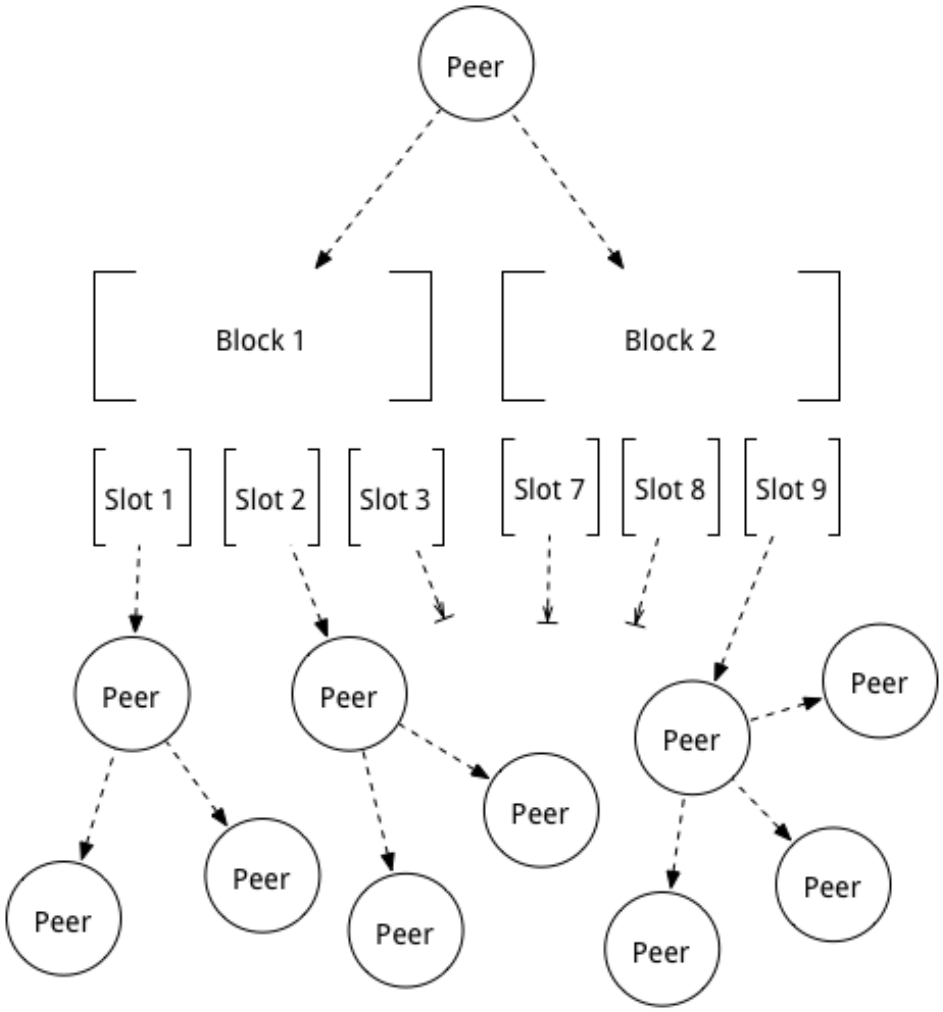
```
 * The message to broadcast to the entire network.
 */
transaction msg;

/**
 * The peers we must broadcast to in order to reach the
 * entire network.
 */
set<peer> peers;

/**
 * Send the message to at least one slot per block.
 */
for (auto & block : blocks)
{
    for (slot in block.slots())
    {
        /**
         * Take a random peer from this slot.
         */
        peers.insert(
            slot.peers()[rand() % (slot.peers().size() - 1)]
        );
    }
}

/**
 * Broadcast the message to all peers on the network.
 */
broadcast(msg, peers);
```

This example algorithm broadcasts a message to a random peer selected from each slot in the system. Each recipient of the message then broadcasts the message to all peers in it's slot. The operation itself is instantaneous reaching all network peers in less than one second. Large scale simulations (100's of millions of nodes) show that the typical operation takes about 300 Milliseconds.



Proof-of-Work Implementation

1. Algorithm

   The chosen algorithm is Whirlpool. The resulting hash consists of splitting 384 bits of whirlpool digest and XORing them together in a way to form 256 bits of output.

   Example:

```
uint256 ret;

uint512 digest = whirlpool(block);

for (uint32 i = 0; i < (sizeof(digest) / 2); i++)
{
    ret[i] =
        digest[i] ^ digest[i + ((sizeof(digest) / 2) / 2)]
    ;
}
```

2. Reward

   The miner reward is adjusted in a way that a majority of all coins will be mined within the first 5 years.

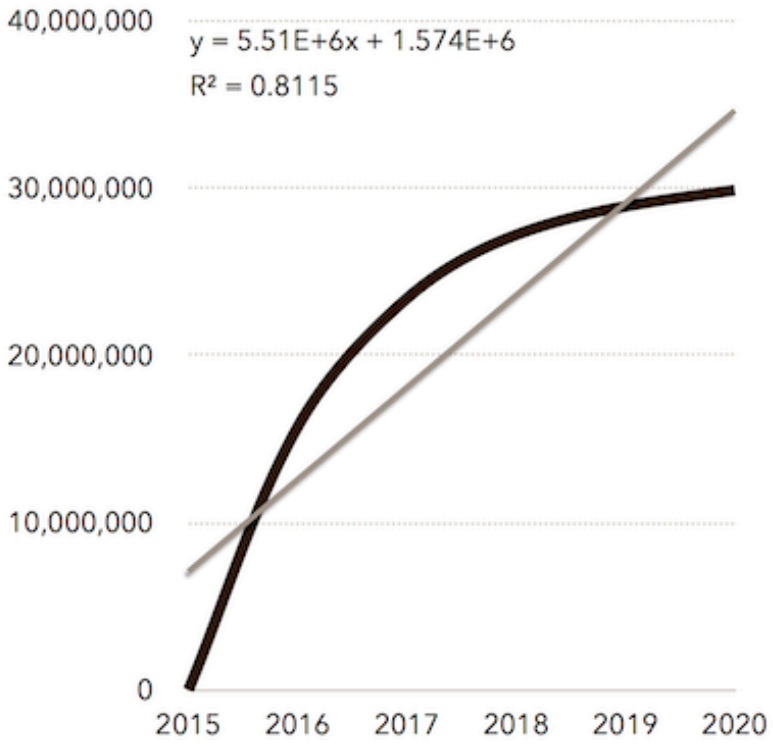
   Example:

```
/**
 * Destroy fees to deflate the money supply.
 */
destroy_fees(fees);

/**
 * Generate the reward.
 */
int64 reward = (1111.0 * (pow((height + 1.0), 2.0))) * 1000

/**
 * Decline the reward every 40000 blocks.
 */
for (auto i = 40000; i <= height; i += 40000)
{
    reward -= reward / 6;
}
```

Money Supply Curve



1. Design Rational

   This is a very efficient way to generate Proof-of-Work while maintaining strong security. The reward model in combination with Proof-of-Stake supports long-term low inflation and energy efficiency.

Proof-of-Stake Implementation

1. Algorithm

   Our implementation is extremely energy efficient, using less than 1% of a single CPU core making it ideal to run in the background or for use on mobile devices. As a result it gives peers the incentive to stay online longer therefore making the network more robust and secure.

2. Design Rational

   This is a very efficient way to generate Proof-of-Stake while promoting the reduction of network churn.

Specifications

- 30.7 million coins.
- Variable block time targeting 80-200 seconds.
- 128 coins per block initially.
- Difficulty is retargeted every block.
- Whirpool Proof-of-Work algorithm.
- 0.7% interest rate by use of energy efficient Proof-of-Stake.
- 0.0005 coin per kilobyte transaction fee.

Summary

It is possible to make cryptographic currencies more performant, energy efficient, secure and private.

Author

John Connor

Public Key:

```
047d3cdc290f94d80ae88fe7457f80090622d064757
9e487a9ad97f77d1c3b3a9e8b596796ebeb23a78214
fc0a95b6a093b3f1d5e2205bd32168ac003f50f4f557
```

Contact:

```
BM-NC49AxAjcqVcF5jNPu85Rb8MJ2d9JqZt
```

References

1. S Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, B. Moeller (2006): Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) (https://tools.ietf.org/html/rfc4492)

2. Dierks T. and Rescorla E. (2008): The Transport Layer Security (TLS) Protocol. (http://tools.ietf.org/html/rfc5246)

3. Wu H. (2004): Stream Cipher HC-256. (http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc256_p3.pdf)

4. Nakamoto S. (2008): Bitcoin: A peer-to-peer electronic cash system. (http://www.bitcoin.org/bitcoin.pdf)

5. King S. and Nadal S. (2012): PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. (http://peercoin.net/assets/paper/peercoin-paper.pdf)

6. The Bitcoin Developers (2013): Satoshi Client Block Exchange. (https://en.bitcoin.it/wiki/Satoshi_Client_Block_Exchange)

7. The Public (2014) : Deep Packet Inspection. (http://en.wikipedia.org/wiki/Deep_packet_inspection)