```
BIP: 33
Layer: Peer Services
Title: Stratized Nodes
Author: Amir Taaki <genjix@riseup.net>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0033
Status: Rejected
Type: Standards Track
Created: 2012-05-15
```

Abstract

As the Bitcoin network scales, roles are fast becoming specialised. In the beginning, a single Bitcoin user would perform the synonymous roles of miner, merchant and end-user. With the growth however of this system, these functions are being abstracted away to specialised services as a natural part of Bitcoin's growth.

Bitcoin's blockchain becomes more unwieldy for end users over time, negatively affecting the usability of Bitcoin clients. As it grows, it becomes ever more impractical to deal with on portable devices or low end machines. Several proposals have been put forward to deal with this such as lightweight (headers-only) clients and skipping validation for blocks before the last checkpoint. However these measures are at best stop-gap workarounds to stave off a growing problem.

This document will examine a proposal which will be termed *stratized nodes*, a modification of an earlier concept termed *blockchain service*.

History

Jan Moller created BCCAPI in 2011. BCCAPI allowed a user's client to delegate blockchain interaction to a remote server. This server would store and manage the blockchain while the user client would run queries against that server.

ThomasV later created Electrum server. BCCAPI's server backend was proprietary, and Electrum required a full Free Software stack. Electrum's server was an adhoc temporary replacement. As it grew and became used, issues started to appear in its design.

Marek Palatinus (slush) drafted a new standard called Stratum to replace Electrum's server. Stratum has multiple transports and is usable as a blockchain server by merchants, miners and user-clients. Electrum moved to using a Stratum implementation first relying on ABE/bitcoind and more recently libbitcoin.

Stratum is unmaintained by Marek Palatinus, suffers from easy resource starvation and denial of service attacks, and is insecure. The proposal specified here is intended to replace the Stratum's role as a blockchain for user-clients. The proposal here is solely concerned with removing the onus of blockchain validation

and lookups from user-clients to specialised services in a secure manner. Any secondary benefits or uses are purely incidental.

Overview

During the initial handshake between Bitcoin nodes, a version packet is sent. version packets have a bitfield called services. Nodes can fill this field to tell the network how they behave and which services they support. NODE_NETWORK (1) means a node can be asked for full blocks for example.

We propose two more values of NODE_SERVICE (2) and NODE_STRATIZED (4).

NODE_SERVICE

- A blockchain service which supports the additional messages "getoutputs" and "getspends".
- Does not respond to "getdata" messages by itself (unless NODE_NETWORK is specified)
- If NODE_NETWORK is specified, then "getdata" for transactions will retrieve them not only from the memory pool but also check the blockchain if necessary.

NODE_STRATIZED

- A node which uses the stratized strategy specified in this document.
- NODE_STRATIZED will relay inventories for accepted transactions.
- Does not support "getblocks" as stratized nodes do not contain the entire blockchain.

Apart from the differences noted above, the nodes are otherwise unchanged in their behaviour from NODE_NETWORK.

Specification

Initialisation

Four new messages are defined which are represented below in C-like pseudocode.

"getoutputs"

```
struct decoded_address
{
    uint8_t payment_type;
    uint8_t address_hash[16];
};

struct get_outputs
{
```

```
    decoded_address dest;
};
```

"outputs"

```
struct point_t
{
    uint8_t hash[32];
    uint32_t index;
};

struct outputs
{
    decoded_address dest;
    uint64_t number_outputs;  // variable uint
    point_t outpoints[];
};
```

"getspend"

```
struct get_spend
{
    point_t outpoint;
};
```

"spend"

```
struct spend
{
    point_t outpoint, inpoint;
};
```

These four messages allow a node to discover the history of a Bitcoin address without needing direct access to the blockchain.

A typical use case might look like:

1. Send "getoutputs" for a decoded Bitcoin address.
2. Receive "outputs", and loop through each contained output point:
   (a) Send "getdata" to download the transaction for that point.
   (b) Send "getspend" for each output point.
3. Receive "spend":
   (a) Send "getdata" to download the transaction for that input point.

This sequence allows the gradual but fast build up of history for an address.

Updates

Nodes receive "inv" messages as normal from service nodes, issuing "getdata" to download the block or transaction data. From this they check for newly sent (in

the input points) or received (in the output points) payments in the transaction data.

Note that blocks must at minimum have their merkle root validated and transactions must be checked for uniqueness by stratized nodes.

Security

The concern here is that stratized nodes are at the mercy of blockchain services. This proposal deals with that issue by designing this protocol in such a way that the implementation can resolve the common history between multiple services.

A stratized node will typically connect to 8 blockchain services. It will only accept an output, spend or inventory vector that has been sent by a common subset of all those services (6 in our example). This spreads the risk between all services, and does not make a node vulnerable to any one rogue blockchain service.

Privacy

The other strategy for thin clients termed *headers-only* or *Simplified. Payment. Verification.* have the same privacy issues as this proposal. SPV resolves this problem by sending out fake requests for transaction data which obfuscates the client data. By sending out a sufficient number of fake requests, privacy can be kept to a sufficient level.

Rationale

NODE_SERVICE does not respond to "getdata" requests by itself (unless used in conjunction with NODE_NETWORK) to prevent starvation attacks. This allows a single trusted NODE_SERVICE architecture (possibly acting as a front-end to multiple backends) to service very many nodes while externalising the costs to the Bitcoin network.

NODE_STRATIZED tries its best to maintain the facade and help upkeep the Bitcoin network (see relaying), but cannot support "getblocks" as it does not have the entire blockchain.

Backwards Compatibility

This proposal is an addon to the current Bitcoin network, and is completely backwards compatible.