```
BIP: 310
Layer: Applications
Title: Stratum protocol extensions
Author: Pavel Moravec <pavel.moravec@braiins.cz>
        Jan Čapek <jan.capek@braiins.cz>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0310
Status: Draft
Type: Informational
Created: 2018-03-10
License: BSD-3-Clause
         CC0-1.0
```

Abstract

This BIP provides a generic mechanism for specifying stratum protocol extensions. At the same time, one of the important extensions that is specified by this BIP is configuration of bits for "version rolling" in nVersion field of bitcoin block header.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Motivation

The initial motivation for specifying some general support for stratum protocol extensions was a need to allow miners to do so called "version rolling", changing value in the first field of the Bitcoin block header.

Version rolling is backwards incompatible change to the stratum protocol because the miner couldn't communicate different block version value to the server in the original version of the stratum protocol. Similarly, a server couldn't communicate safe bits for rolling to a miner. So both miners and pools need to implement some protocol extension to support version rolling.

Typically, if a miner sends an unknown message to a server, the server closes the connection (not all implementations do that but some do). So it is not very safe to try to send unknown messages to servers.

We can use this opportunity to make one backwards incompatible change to the protocol to support multiple extensions in the future. In a way that a miner can advertise its capabilities and at the same time it can request some needed features from the server.

It is preferable that the same mechanism for feature negotiation can be used for not yet known features. It SHOULD be easy to implement in the mining software too.

We introduce one new message to the stratum protocol (**"mining.configure"**) which handles the initial configuration/negotiation of features in a generic way. So that adding features in the future can be done without a necessity to add new messages to stratum protocol.

Each extension has its unique string name, so called **extension code**.

Specification

Currently, the following extensions are defined:

- **"version-rolling"**
- **"minimum-difficulty"**
- **"subscribe-extranonce"**

Additional data types

The following names are used as type aliases, making the message description easier.

- **TMask** - case independent hexadecimal string of length 8, encoding an unsigned 32-bit integer (~`[0-9a-fA-F]{8}`)
- **TExtensionCode** - non-empty string with a value equal to the name of some protocol extension.
- **TExtensionResult** - `true` / `false` / *String*.
  - `true` = The requested feature is supported and its configuration understood and applied.
  - `false` = The feature is not supported or unknown.
  - *String* = Error message containing information about what went wrong.

Request "mining.configure"

This message (JSON RPC Request) SHOULD be the **first message** sent by the miner after the connection with the server is established. The client uses the message to advertise its features and to request/allow some protocol extensions.

The reason for it being the first is that we want the implementation and possible interactions to be as easy and simple as possible. An extension can define explicitly what does a repeated configuration of that extension mean.

Each extension code provides a namespace for its extension parameters and extension return values. By convention, the names are formed from extension codes by adding "." and a parameter name. The same applies for the return values, which are transferred in a result map too. E.g. "version-rolling.mask" is the name of the parameter "mask" of extension "version-rolling".

Parameters:

- **extensions** (REQUIRED, List of *TExtensionCode*)

- Each string in the list MUST be a valid extension code. The meaning of each code is described independently as part of the extension definition. A miner SHOULD advertise all its available features.

- **extension-parameters** (REQUIRED, *Map of (String -> Any)*)

- Parameters of the requested/allowed extensions from the first parameter.

Return value:

- *Map of (String -> Any)*

- Each code from the **extensions** list MUST have a defined return value (*TExtensionCode* -> *TExtensionResult*). This way the miner knows if the extension is activated or not. E.g. `{"version-rolling":false}` for unsupported version rolling.
- Some extensions need additional information to be delivered to the miner. The return value map is used for this purpose.

Example request (new-lines added):

```
{"method": "mining.configure",
 "id": 1,
 "params": [["minimum-difficulty", "version-rolling"],
        {"minimum-difficulty.value": 2048,
         "version-rolling.mask": "1fffe000", "version-rolling.min-bit-count": 2}]}
```

(The miner requests extensions `"version-rolling"` and `"minimum-difficulty"`. It sets the parameters according to the extensions' definitions.)

Example result (new-lines added):

```
{"error": null,
 "id": 1,
 "result": {"version-rolling": true,
        "version-rolling.mask": "18000000",
        "minimum-difficulty": true}}
```

Defined extensions

Extension "version-rolling"

This extension allows the miner to change the value of some bits in the version field in the block header. Currently there are no standard bits used for version rolling so they need to be negotiated between a miner and a server.

A miner sends the server a mask describing bits which the miner is capable of changing. 1 = changeable bit, 0 = not changeable (`miner_mask`) and a minimum number of bits that it needs for efficient version rolling.

A server typically allows you to change only some of the version bits (`server_mask`) and the rest of the version bits are fixed. E.g. because the block needs to be valid or some signaling is in place.

The server responds to the configuration message by sending a mask with common bits intersection of the miner's mask and its a mask (`response = server_mask & miner_mask`)

Example request (a miner capable of changing any 2 bits from a 16-bit mask):

```
{"method": "mining.configure", "id": 1, "params": [["version-rolling"], {"version-rolling.ma
```

Example result (success):

```
{"error": null, "id": 1, "result": {"version-rolling": true, "version-rolling.mask": "18000
```

Example result (unknown extension):

```
{"error": null, "id": 1, "result": {"version-rolling": false}}
```

Extension parameters:

- **"version-rolling.mask"** (OPTIONAL, *TMask*, default value `"ffffffff"`)

- Bits set to 1 can be changed by the miner. This value is expected

to be stable for the whole mining session. A miner doesn't have to send the mask, in this case a default full mask is used.

Extension return values:

- **"version-rolling"** (REQUIRED, *TExtensionResult*)

- When responded with `true`, the server will accept new parameter of **"mining.submit"**, see later.

- **"version-rolling.mask"** (REQUIRED, *TMask*)

- Bits set to 1 are allowed to be changed by the miner. If a miner changes bits with mask value 0, the server will reject the submit.
  - The server SHOULD return the largest mask possible (as many bits set to 1 as possible). This can be useful in a mining proxy setup when a proxy needs to negotiate the best mask for its future clients. There is a [Draft BIP](https://github.com/bitcoin/bips/pull/661/files) describing available nVersion bits. The server SHOULD pick a mask that preferably covers all bits specified in the BIP.

- **"version-rolling.min-bit-count"** (REQUIRED, *TMask*)

- The miner also provides a minimum number of bits that it needs for efficient version rolling in hardware. Note that this parameter provides important diagnostic information to the pool server. If the requested bit count exceeds the limit of the pool server, the miner always has the chance to operate in a degraded mode without using full hashing power. The pool server SHOULD NOT terminate miner connection if this rare mismatch case occurs.

Notification "mining.set_version_mask"

Server notifies the miner about a new mask valid for the connection. This message can be sent at any time after the successful setup of the version rolling extension by the "mining.configure" message. The new mask is valid **immediately**, so that the server doesn't wait for the next job.

Parameters:

- *mask* (REQUIRED, *TMask*): The meaning is the same as the **"version-rolling.mask"** return parameter.

Example:

```
{"params":["00003000"], "id":null, "method": "mining.set_version_mask"}
```

Changes in request "mining.submit"

Immediately after successful activation of the version-rolling extension (result to **"mining.configure"** sent by server), the server MUST accept an additional parameter of the message **"mining.submit"**. The client MUST send one additional parameter, **version_bits** (6th parameter, after *worker_name*, *job_id*, *extranonce2*, *ntime* and *nonce*).

Additional parameters:

- *version_bits* (REQUIRED, *TMask*) - Version bits set by miner.

- Miner can set only bits corresponding to the set bits in the last received mask from the server either as response to "mining.configure" or "mining.set_version_mask" notification (`last_mask`). This must hold:

```
version_bits & ~last_mask ==  0
```

- The server computes *nVersion* for the submit as follows:

```
nVersion = (job_version & ~last_mask) | (version_bits & last_mask)
```

where `job_version` is the block version sent to miner as part of job with id `job_id`.

Extension "minimum-difficulty"

This extension allows miner to request a minimum difficulty for the connected machine. It solves a problem in the original stratum protocol where there is no way how to communicate hard limit of the connected device.

Extension parameters:

- **"minimum-difficulty.value"** (REQUIRED, *Integer/Float*, >= 0)

- The minimum difficulty value acceptable for the miner/connection. The value can be 0 for essentially disabling the feature.

Extension return values:

- **"minimum-difficulty"** (REQUIRED, *TExtensionResult*)

- Whether the minimum difficulty was accepted or not.
    - This extension can be configured multiple times by calling "mining.configure" with "minimum-difficulty" code again.

Extension "subscribe-extranonce"

Parameter-less extension. Miner advertises its capability of receiving message **"mining.set_extranonce"** message (useful for hash rate routing scenarios).

Extension "info"

Miner provides additional text-based information.

Extension parameters:

- **"info.connection-url"** (OPTIONAL, *String*)

- Exact URL used by the mining software to connect to the stratum server.

- **"info.hw-version"** (OPTIONAL, *String*)

- Manufacturer specific hardware revision string.

- **"info.sw-version"** (OPTIONAL, *String*)

- Manufacturer specific software version

- **"info.hw-id"** (OPTIONAL, *String*)

- Unique identifier of the mining device

Compatibility

Currently, there is a similar protocol feature **mining.capabilities** that was intended for various protocol extensions. However, **mining.configure** is incompatible with this feature as it requires a server response confirming all accepted/negotatied extensions. The reason why we made it incompatible is that **mining.capabilities** request has no associated response.

Copyright

This document is dual licensed as BSD 3-clause, and Creative Commons CC0 1.0 Universal.