STRATUM FOR ZCASH

April 3, 2017

Abstract

This ZIP describes the Zcash variant of the Stratum protocol, used by miners to communicate with mining pool servers.

This document follows the common conventions defined for ZIPs in [ZIP-1]_.

Motivation

Many existing cryptocurrency miners and pools use the original Stratum protocol [Slushpool-Stratum]_ [Bitcointalk-Stratum]_ for communication, in situations where the miner does not require any control over what they mine (for example, a miner connected to a local [P2Pool]_ node). However, the protocol is very specific to Bitcoin, in that it makes assumptions about the block header format, and the available nonce space [Bitcoin-Block]_. Zcash has made changes that invalidate these assumptions.

Having a formal specification for a Zcash-compatible Stratum-style mining protocol means that existing pool operators and miner authors can quickly and easily migrate their frameworks to the Zcash network, with no ambiguity about interoperability.

Specification

The Stratum protocol is an instance of [JSON-RPC-1.0]_. The miner is a JSON-RPC client, and the Stratum server is a JSON-RPC server. The miner starts a session by opening a standard TCP connection to the server, which is then used for two-way line-based communication:

- The miner can send requests to the server.

- The server can respond to requests.
- The server can send notifications to the client.

All communication for a particular session happens through a single connection, which is kept open for the duration of the session. If the connection is broken or either party disconnects, the active session is ended. Servers MAY support session resuming; this is negotiated between the client and server during initial setup (see Session Resuming_).

Each request or response is a JSON string, terminated by an ASCII LF character (denoted in the rest of this specification by `\n`). The LF character MUST NOT appear elsewhere in a request or response. Client and server implementations MAY assume that once they read a LF character, the current message has been completely received.

Per [JSON-RPC-1.0]_, there is no requirement for the `id` property in requests and responses to be unique; only that servers MUST set `id` in their responses equal to that in the request they are responding to (or `null` for notifications). However, it is RECOMMENDED that clients use unique ids for their requests, to simplify their response parsing.

In the protocol messages below, `(content)` indicates that `content` is optional. Variable names are indicated in *EMPHASIS*. All other characters are part of the protocol message.

Error Objects

The [JSON-RPC-1.0]_ specification allows for error objects in responses, but does not specify their format. The original Stratum protocol uses the following format for error responses [Slushpool-Stratum]_:

> {"id": ##, "result": null, "error": [*ERROR_CODE*, "*ERROR_MESSAGE*", *TRACEBACK*]} `\n`

For compatibility, this format is retained. We therefore define an error object as an array:

> [*ERROR_CODE*, "*ERROR_MESSAGE*", *TRACEBACK*]

`ERROR_CODE` (int) Indicates the type of error that occurred.

The error codes are to be interpreted as described in [JSON-RPC-2.0]_. The following application error codes are defined:

- 20 - Other/Unknown
- 21 - Job not found (=stale)

- 22 - Duplicate share
- 23 - Low difficulty share
- 24 - Unauthorized worker
- 25 - Not subscribed

`ERROR_MESSAGE` (str) A human-readable error message. The message SHOULD be limited to a concise single sentence.

`TRACEBACK` Additional information for debugging errors. Format is server-specific.

Miners MAY attempt to parse the field for displaying to the user, and SHOULD fall back to rendering it as a JSON string.

Servers MUST set this to `null` if they have no additional information.

Miners SHOULD display a human-readable message to the user. This message can be derived from either `ERROR_CODE` or `ERROR_MESSAGE`, or both. An example of using `ERROR_CODE` over `ERROR_MESSAGE` might be that the miner UI offers localisation.

Protocol Flow

- Client sends `mining.subscribe` to set up the session.
- Server replies with the session information.
- Client sends `mining.authorize` for their worker(s).
- Server replies with the result of authorization.
- Server sends `mining.set_target`.
- Server sends `mining.notify` with a new job.
- Client mines on that job.
- Client sends `mining.submit` for each solution found.
- Server replies with whether the solution was accepted.
- Server sends `mining.notify` again when there is a new job.

Nonce Parts

In Bitcoin, blocks contain two nonces: the 4-byte block header nonce, and an extra nonce in the coinbase transaction [Bitcoin-Block]_. The original Stratum protocol splits this extra nonce into two parts: one set by the server (used for splitting the search space amongst connected miners), and the other iterated by the miner [Slushpool-Stratum]_. The nonce in Zcash's block header is 32 bytes long [Zcash-Block]_, and thus can serve both purposes simultaneously.

We define two nonce parts:

`NONCE_1` The server MUST pick such that `len(NONCE_1) < 32` in bytes.

`NONCE_2` The miner MUST pick such that `len(NONCE_2) = 32 - len(NONCE_1)` in bytes.

In hex, `lenHex(NONCE_2) = 64 - lenHex(NONCE_1)`, and both lengths are even.

The nonce in the block header is the concatenation of `NONCE_1` and `NONCE_2` in hex. This means that miner using bignum representations of nonce MUST increment by `1 << len(NONCE_1)` to avoid altering `NONCE_1` (because the encoding of nonce in the block header is little endian, in line with the other 32-byte fields [Bitcoin-Block]_ [Zcash-Block]_).

Session Resuming

Servers that support session resuming identify this by setting a `SESSION_ID` in their initial response. Servers MAY set `SESSION_ID` to `null` to indicate that they do not support session resuming. Servers that do not set `SESSION_ID` to `null` MUST cache the following information:

- The session ID.
- `NONCE_1`
- Any active job IDs.

Servers MAY drop entries from the cache on their own schedule.

When a miner connects using a previous `SESSION_ID`:

- If the cache contains the `SESSION_ID`, the server's initial response MUST be constructed from the cached information.
- If the server does not recognise the session, the `SESSION_ID` in the server's initial response MUST NOT equal the `SESSION_ID` provided by the miner.

Miners MUST re-authorize all workers upon resuming a session.

Methods

`mining.subscribe()`

Request:

> {"id": 1, "method": "mining.subscribe", "params": ["*MINER_USER_AGENT*", "*SESSION_ID*", "*CONNECT_HOST*", *CONNECT_PORT*]} `\n`

`MINER_USER_AGENT` **(str)** A free-form string specifying the type and version of the mining software. Recommended syntax is the User Agent format used by Zcash nodes.

Example: `MagicBean/1.0.0`

`SESSION_ID` **(str)** The id for a previous session that the miner wants to resume (e.g. after a temporary network disconnection) (see Session Resuming_).

MAY be `null` indicating that the miner wants to start a new session.

`CONNECT_HOST` **(str)** The host that the miner is connecting to (from the server URL).

Example: `pool.example.com`

`CONNECT_PORT` **(int)** The port that the miner is connecting to (from the server URL).

Example: `3337`

Response:

{"id": 1, "result": ["*SESSION_ID*", "*NONCE_1*"], "error": null}
`\n`

`SESSION_ID` **(str)** The session id, for use when resuming (see Session Resuming_).

`NONCE_1` **(hex)** The first part of the block header nonce (see Nonce Parts_).

`mining.authorize()`

A miner MUST authorize a worker in order to submit solutions. A miner MAY authorize multiple workers in the same session; this could be for statistical purposes on the particular server being used. Details of such purposes are outside the scope of this specification.

Request:

{"id": 2, "method": "mining.authorize", "params": ["*WORKER_NAME*", "*WORKER_PASSWORD*"]} `\n`

`WORKER_NAME` **(str)** The worker name.

`WORKER_PASSWORD` **(str)** The worker name.

Response:

{“id”: 2, “result”: *AUTHORIZED*, “error”: *ERROR*} `\n`

`AUTHORIZED` **(bool)** MUST be `true` if authorization succeeded. Per [JSON-RPC-1.0]_, MUST be `null` if there was an error.

`ERROR` **(obj)** An error object. MUST be `null` if authorization succeeded.

If authorization failed, the server MUST provide an error object describing the reason. See Error Objects_ for the object format.

`mining.set_target()`

Server message:

{“id”: null, “method”: “mining.set_target”, “params”: [“*TARGET*”]} `\n`

`TARGET` **(hex)** The server target for the next received job and all subsequent jobs (until the next time this message is sent). The miner compares proposed block hashes with this target as a 256-bit big-endian integer, and valid blocks MUST NOT have hashes larger than (above) the current target (in accordance with the Zcash network consensus rules [Zcash-Target]_).

Miners SHOULD NOT submit work above this target. Miners SHOULD validate their solutions before submission (to avoid both unnecessary network traffic and wasted miner time).

Servers MUST NOT accept submissions above this target for jobs sent after this message. Servers MAY accept submissions above this target for jobs sent before this message, but MUST check them against the previous target.

When displaying the current target in the UI to users, miners MAY convert the target to an integer difficulty as used in Bitcoin miners. When doing so, miners SHOULD use `powLimit` (as defined in `src/chainparams.cpp`) as the basis for conversion.

`mining.notify()`

Server message:

{“id”: null, “method”: “mining.notify”, “params”: [“*JOB_ID*”, “*VERSION*”, “*PREVHASH*”, “*MERKLEROOT*”, “*RESERVED*”, “*TIME*”, “*BITS*”, *CLEAN_JOBS*]} `\n`

`JOB_ID` **(str)** The id of this job.

`VERSION` **(hex)** The block header version, encoded as in a block header (little-endian `int32_t`).

Used as a switch for subsequent parameters. At time of writing, the only defined block header version is 4. Miners SHOULD alert the user upon receiving jobs containing block header versions they do not know about or support, and MUST ignore such jobs.

Example: `04000000`

The following parameters are only valid for `VERSION == "04000000"`:

`PREVHASH` **(hex)** The 32-byte hash of the previous block, encoded as in a block header.

`MERKLEROOT` **(hex)** The 32-byte Merkle root of the transactions in this block, encoded as in a block header.

`RESERVED` **(hex)** A 32-byte reserved field, encoded as in a block header. Zero by convention (in hex, `0000000000000000000000000000000000000000000000000000000000000000`).

`TIME` **(hex)** The block time suggested by the server, encoded as in a block header.

`BITS` **(hex)** The current network difficulty target, represented in compact format, encoded as in a block header.

`CLEAN_JOBS` **(bool)** If true, a new block has arrived. The miner SHOULD abandon all previous jobs.

`mining.submit()`

Request:

{“id”: 4, “method”: “mining.submit”, “params”: [“*WORKER_NAME*”, “*JOB_ID*”, “*TIME*”, “*NONCE_2*”, “*EQUIHASH_SOLUTION*”]}
`\n`

`WORKER_NAME` **(str)** A previously-authenticated worker name.

Servers MUST NOT accept submissions from unauthenticated workers.

`JOB_ID` **(str)** The id of the job this submission is for.

Miners MAY make multiple submissions for a single job id.

`TIME` **(hex)** The block time used in the submission, encoded as in a block header.

MAY be enforced by the server to be unchanged.

`NONCE_2` **(hex)** The second part of the block header nonce (see Nonce Parts_).

`EQUIHASH_SOLUTION` **(hex)** The Equihash solution, encoded as in a block header (including the compactSize at the beginning in canonical form [Bitcoin-CompactSize]_).

Result:

{"id": 4, "result": *ACCEPTED*, "error": *ERROR*} `\n`

`ACCEPTED` **(bool)** MUST be `true` if the submission was accepted. Per [JSON-RPC-1.0]_, MUST be `null` if there was an error.

`ERROR` **(obj)** An error object. Per [JSON-RPC-1.0]_, MUST be `null` if the submission was accepted without error.

If the submission was not accepted, the server MUST provide an error object describing the reason for not accepting the submission. See Error Objects_ for the object format.

`client.reconnect()`

Server message:

{"id": null, "method": "client.reconnect", "params": [("*HOST*", *PORT*, *WAIT_TIME*)]} `\n`

`HOST` **(str)** The host to reconnect to.

Example: `pool.example.com`

`PORT` **(int)** The port to reconnect to.

Example: `3337`

`WAIT_TIME` **(int)** Time in seconds that the miner should wait before reconnecting.

If `client.reconnect` is sent with an empty parameter array, the miner SHOULD reconnect to the same host and port it is currently connected to.

`mining.suggest_target()`

Request (optional):

> {"id": 3, "method": "mining.suggest_target", "params": ["*TARGET*"]} `\n`

`TARGET` (hex) The target suggested by the miner for the next received job and all subsequent jobs (until the next time this message is sent).

The server SHOULD reply with `mining.set_target`. The server MAY set the result id equal to the request id.

Rationale

Why does `mining.subscribe` include the host and port?

- It has the same use cases as the `Host:` header in HTTP. Specifically, it enables virtual hosting, where virtual pools or private URLs might be used for DDoS protection, but that are aggregated on Stratum server backends. As with HTTP, the server CANNOT trust the host string.
- The port is included separately to parallel the `client.reconnect` method; both are extracted from the server URL that the miner is connecting to (e.g. `stratum+tcp://pool.example.com:3337`).

Why use the 256-bit target instead of a numerical difficulty?

- There is no protocol ambiguity when using a target. A server can pick a specific target (by whatever algorithm), and enforce it cleanly on submitted jobs.
  - A numerical difficulty must be converted into a target by miners, which adds unnecessary complexity, results in a loss of precision, and leaves ambiguity over the conversion and the validity of resulting submissions.
- The minimum numerical difficulty in Bitcoin's Stratum protocol is 1, which corresponds to `powLimit`. This makes it harder to test miners and servers. A target can represent difficulties lower than the minimum.

Does a 256-bit target waste bandwidth?

- The target is generally not set as often as solutions are submitted, so any effect is minimal.

- Zcash's proof-of-work, Equihash, is much slower than Bitcoin's, so any latency caused by the size of the target is minimal compared to the overall solver time.
- For the current Equihash parameters (200/9), the Equihash solution dominates bandwidth usage.

Why does `mining.submit` include `WORKER_NAME`?

- `WORKER_NAME` is only included here for statistical purposes (like monitoring performance and/or downtime). `JOB_ID` is used for pairing server-stored jobs with submissions.

Reference Implementation

- str4d's standalone miner

Acknowledgements

Thanks to:

- 5a1t for the initial brainstorming session.
- Daira Hopwood for her input on API selection and design.
- Marek Palatinus (slush) and his colleagues for their refinements, suggestions, and robust discussion.
- Jelle Bourdeaud'hui (razakal) and ocminer for their help with testing and finding implementation bugs in the specification.

This ZIP was edited by Daira Hopwood.

References