Polkadot Networking

Protocol Specification

Contents

0.1 Introduction

The Polkadot network is decentralized and does not rely on any central authority or entity in order to achieve a its fullest potential of provided functionality. Each node with the network can authenticate itself and its peers by using cryptographic keys, including establishing fully encrypted connections. The networking protocol is based on the open and standardized `libp2p` protocol, including the usage of the distributed Kademlia hash table for peer discovery.

0.1.1 Discovery mechanism

The Polkadot Host uses varies mechanism to find peers within the network, to establish and maintain a list of peers and to share that list with other peers from the network.

The Polkadot Host uses various mechanism for peer dicovery.

- Bootstrap nodes - hard-coded node identities and addresses provided by network configuration itself. Those addresses are selected an updated by the developers of the Polkadot Host. Node addresses should be selected based on a reputation metric, such as reliability and uptime.
- mDNS - performs a broadcast to the local network. Nodes that might be listing can respond the the broadcast.
- Kademlia requests - Kademlia supports `FIND_NODE` requests, where nodes respond with their list of available peers.

0.1.2 Connection establishment

The Polkadot Host can establish a connection with any peer it knows the address. `libp2p` uses the `multistream-select` protocol in order to establish an encryption and multiplexing layer. The Polkadot Host supports multiple base-layer protocols:

- TCP/IP - addresses in the form of `/ip4/1.2.3.4/tcp/` establish a TCP connection and negotiate a encryption and multiplexing layer.
- Websockets - addresses in the form of `/ip4/1.2.3.4/ws/` establish a TCP connection and negotiate the Websocket protocol within the connection. Additionally, a encryption and multiplexing layer is negotiated within the Websocket connection.
- DNS - addresses in form of `/dns/website.domain/tcp/` and `/dns/website.domain/ws/`.

After a base-layer protocol is established, the Polkadot Host will apply the Noise protocol.

0.1.3 Substreams

After the node establishes a connection with a peer, the use of multiplexing allows the Polkadot Host to open substreams. Substreams allow the negotiation of *application-specific protocols*, where each protocol servers a specific utility.

The Polkadot Host adoptes the following, standardized `libp2p` application-specific protocols:

- `/ipfs/ping/1.0.0` - Open a substream to a peer and initialize a ping to verify if a connection is till alive. If the peer does not respond, the connection is dropped.
- `/ipfs/id/1.0.0` - Open a substream to a peer to ask information about that peer.
- `/<protocol_id>/kad/` - Open a substream for Kademlia `FIND_NODE` requests.

Additional, non-standardized protocols:

- `/<protocol-id>/sync/2` - a request and response protocol that allows the Polkadot Host to perform information about blocks.
- `/<protocol-id>/light/2` - a request and response protocol that allows a light client to perform information about the state.
- `/<protocol-id>/transactions/1` - a notification protocol which sends transactions to connected peers.
- `/<protocol-id>/block-announces/1` - a notification protocol which sends blocks to connected peers.

0.2 Network Messages

Definition 1 *The roles type, R, is a 8-bit unsigned integer indicating the role of the node. Following options are available: TODO: verify bitmask*

- `0`*: no network.*
- `1`*: full node, does not participate in consensus.*
- `2`*: light client node.*
- `4`*: authory node.*

Definition 2 *The block attributes, B_{attr}, is a 8-bit unsigned integer indicating the artifacts to request. TODO: verify bitmask*

- `1`*: include block header.*
- `2`*: include block body*
- `4`*: include block receipt*
- `8`*: include block message queue.*
- `16`*: include block justification.*

Definition 3 *Block enumeration direction is a varying data type of the following values:*

$$S_d = \begin{cases} 0, & \textit{enumerate in ascending order (from child to parent)} \\ 1, & \textit{enumerate in descending order (from parent to canonical child} \end{cases}$$

Definition 4 *Block state in the chain is a varying data type of the following values:*

$$S_b = \begin{cases} 0, & \textit{Block is not part of the best chain} \\ 1, & \textit{Latest best block} \end{cases}$$

Definition 5 *A proof that some set of key-value pairs are included in the storage trie. The proof contains the storage values so that the partial storage backend can be reconstructed by a verifier that does not already have access to the key-value pairs.*

The proof consists of the set of serialized nodes in the storage trie accessed when looking up the keys covered by the proof. Verifying the proof requires constructing the partial trie from the serialized nodes and performing the key lookups

$$S_p := (A_0, ...A_n)$$

where A is a byte array containing the trie proofs.

0.2.1 Status Packets

A status message sent on connection.

$$M^S := (P_v, Min_v, R, H_i(B), H_h(B), H_h(B_{gen}))$$

where each value represents:

- P_v: a UINT32 indicating the Protocol version.
- Min_v: a UINT32 indicating the minimum supported version.
- R: Supported roles (Def. 1).
- $H_i(B)$: a UINT32 indicating the best block number.
- $H_h(B)$: a 32-byte array indicating the best block hash.
- $H_h(B_{gen})$: a 32-byte array indicating the genesis block hash.

0.2.2 Block Request

Request blocks from a peer.

$$M_R^B eq := (R_{id}, B_{attr}, H_h(B_s), Option(H_h(B_e)), S_d, Option(|\mathbb{B}|))$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- B_{attr}: block attributes to request (Def. 2).
- $H_h(B_s)$: indicates the starting point from this block, which can either be 32-byte block hash or a UINT32 block number.

- $Option(H_h(B_e))$: indicates the ending point to the 32-byte block hash. An implementation defined maximum is used when unspecified.
- S_d: sequence direction (Def. 3).
- $Option(|A|)$: maximum number of blocks to return. An implementation defined maximum is used when unspecified.

0.2.3 Block Response

Response to block request (Sect. 0.2.2).

$$M_{Res}^{B} := (R_{id}, (B_0^{data}, ..., B_n^{data}))$$

where each value represents:

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- B_n^{data}: the block data (Def. 6)

Definition 6 *The block data sent in response.*

$$\begin{aligned} B^{data} &:= (P^1, P^2) \\ P^1 &:= (H_h(B), Option(B_{head}), Option(B_{body})) \\ P^1 &:= (Option(B_{reci}), Option(B_{msg}), Option(B_{just})) \end{aligned}$$

TODO: @fabio: define/link those types

- $H_h(B)$: *the 32-byte block hash.*
- $Option(B_{head})$: *the block header, if requested.*
- $Option(B_{body})$: *the block body, if requested.*
- $Option(B_{reci})$: *the block receipt, if requested.*
- $Option(B_{msg})$: *the block message, if requested.*
- $Option(B_{just})$: *the justification, if requested.*

0.2.4 Block Announce

Announce a new complete relay chain block on the network.

$$M_A^B := (B_{head}, Option(S_b), Option(A))$$

where each value represents:

- H_b: the new block header.

- $Option(S_b)$: block state (Def. 4).
- $Option(A)$: a byte array containing the data associated with this block announcement. For example a candidate message.

0.2.5 Transactions

A collection of transactions, where each transaction is a byte array.

$$M^{Tx} := (A_0, ...A_n)$$

0.2.6 Consensus Protocol

A consensus message.

$$M_P^C := (e_{id}, A)$$

where each value represents:

- e_{id}: a 4-byte unique ID of the consensus engine.
- A: a byte array containing the message payload.

0.2.7 Remote Method Call Request

Remote call request.

$$M_{Req}^{RMC} := (R_{id}, H_h(B), A^1, A^2)$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_h(B)$: a 32-byte block hash at which to perform this call.
- A^1: a byte array containing a valid UTF-8 sequence indicating the method name.
- A^2: a byte array containing the call data.

0.2.8 Remote Method Call Response

Remote call response.

$$M_{Res}^{RMC} := (R_{id}, S_p)$$

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- S_p: a storage proof (Def. 5).

0.2.9 Remote Storage Read Request

Remote storage read request.

$$M_{Req}^{RSR} := (R_{id}, H_h(B), (A_0, ..., A_n))$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_h(B)$: a 32-byte block hash at which to perform this call.
- A: a byte array containing the storage key.

0.2.10 Remote Storage Read Response

Remote storage read response.

$$M_{Res}^{RSR} := (R_{id}, S_p)$$

where each value represents:

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- S_p: a storage proof (Def. 5).

0.2.11 Remote Header Request

Remote header request.

$$M_{Req}^{RHR} := (R_{id}, H_i(B))$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_i(B)$: a UINT32 indicating the block number the header is for.

0.2.12 Remote Header Response

$$M_{Res}^{RHR} := (R_{id}, Option(B_{head}), S_p)$$

where each value represents:

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- $Option(B_{head})$: the block header. This is `None` if proof generation has failed (e.g. header is unknown)
- S_p: a storage proof (Def. 5).

0.2.13 Remote Changes Request

Remote changes request.

$$M_{Req}^{RCR} := (R_{id}, H_h(B_s), H_h(B_e), B_{min}, B_{max}, Option(A^1), A^2)$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_h(B_s)$: a 32-byte block hash indicating the starting point.
- $H_h(B_e)$: . a 32-byte block hash indicating the ending point.
- B_{min}: a 32-byte block hash indicating the first block for which the requester has the changes trie root. All other affected roots must be proved.
- B_{max}: a 32-byte block hash indicating the last block that we can use when querying changes.
- $Option(A^1)$: a byte array containing the child storage key which changes are requested for.
- A^2: a byte array containing the storage key which changes are requested for.

0.2.14 Remote Changes Response

Remote changes response.

$$\begin{aligned} M_{Res}^{RCR} &:= (R_{id}, B_{max}, C_p, (R_0, ..., R_n), S_p) \\ C_p &:= (A_0, ..., A_n) \\ R &:= (UINT32, H(N)) \end{aligned}$$

where each value represents:

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- max: a UINT32 indicating the max block number the proof has been generated with. Should be less than or equal to the block number of B_{max} as defined in Section 0.2.13.
- C_p: a byte array containing changes proofs.
- A: a byte array containing the 32-byte proof. The byte array does not have a length prefix.
- R: changes trie root missing on the requesters node. TODO: @fabio
- S_p: missing changes tries roots proof (Def. 5).

0.2.15 Remote Child Storage Read Request

Remote storage read child request.

$$M_{Req}^{RCSR} := (R_{id}, H_h(B), A^1, A^2)$$

where each value represents:

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_h(B)$: a 32-byte block hash at which to perform the call.
- A^1: byte array containing the child storage key.
- A^2: byte array containing the storage key.

0.2.16 Remote Child Storage Read Response

TODO: Seems to be missing in the code? Unless another type gets reused.

0.2.17 Finality Proof Request

Finality proof request.

$$M_{Req}^{FP} := (R_{id}, H_h(B), A)$$

- R_{id}: a UINT64 indicating the unique ID of a request.
- $H_h(B)$: a 32-byte hash of the block to request proof for.
- A: a byte array containing additional data required for proving finality.

0.2.18 Finality Proof Response

Finality proof response.

$$M_{Res}^{FP} := (R_{id}, H_h(B), Option(A_0, ..., A_n))$$

where each value represents:

- R_{id}: a UINT64 indicating the ID of the request this response was made for.
- $H_h(B)$: a 32-byte hash of the block (the same hash as specified in the request (Sect. 0.2.17))
- A: a byte array containing the finality proof, if available.

0.2.19 Batch of Conensus Protocol Messages

$$bM_P^C := (M_{P0}^C, ..., M_{P1}^C)$$

bM_P^C is an array containing multiple consensus messages (Sect. 0.2.6)