Polkadot RE Spec

November 23, 2018

1 Conventions and Definitions

Definition 1. ***Runtime*** *is the state transition function of the decentralized ledger protocol.*

Definition 2. *A* ***path graph*** *or a* ***path*** *of n nodes, formally referred to as* $\boldsymbol{P_n}$*, is a tree a with two nodes of vertex degree 1 and the other n-2 nodes of vertex degree 2. Therefore, $P_n$can be represented by sequences of $(v_1, \ldots, v_n)$ where $e_i = (v_i, v_{i+1})$ for $1 \leqslant i \leqslant n - 1$ is the edge which connect v_i and v_{i+1}.*

Definition 3. ***radix r tree*** *is a variant of a trie in which:*

- *Every node has at most r children where $r = 2^x$ for some x.*
- *Each node that is the only child of a parent, which does not represent a valid key is merged with its parent.*

As a result, in a radix tree, any path whose interior vertices all have only one child and does not represent a valid key in the data set, is compressed into a single edge. This improves space efficiency when the key space is sparse.

Definition 4. *By a* ***sequences of bytes*** *or a* ***byte array****, b, of length n, we refer to*

$$b := (b_0, b_1, \ldots, b_{n-1}) \text{ such that } 0 \leqslant b_i \leqslant 255$$

We define $\mathbb{B}_n$ to be the ***set of all byte arrays of length n****. Furthermore, we define:*

$$\mathbb{B} := \bigcup_{i=0}^{\infty} \mathbb{B}_i$$

Notation 5. *We represent the concatination of byte arrays $a := (a_0, \ldots, a_n)$ and $b := (b_0, \ldots, b_m)$ by:*

$$a || b := (a_0, \ldots, a_n, b_0, \ldots, b_m)$$

Definition 6. *For a given byte b the **bitwise representation** of b is defined as*

$$b := b^7 \dots b^0$$

where

$$b = 2^0\,b^0 + 2^1\,b^1 + \cdots + 2^7\,b^7$$

2 Block

In Polkadot RE, a block is made of two main parts, namely the *block header* and the *list of extrinsics*. *The Extrinsics* represent the generalization of the concept of *transaction*, containing any set of data that is external to the system, and which the underlying chain wishes to validate and keep track of.

2.1 Block Header

The block header is designed to be minimalistic in order to boost the efficiency of the light clients. It is defined formally as follows:

Definition 7. *The header of block B,* Head(B) *is a 5-tuple containing the following elements:*

- ***parent_hash:*** *is the 32-byte Blake2s hash of the header of the parent of the block indicated henceforth by* $\boldsymbol{H_p}$.
- ***number:*** *formally indicated as* $\boldsymbol{H_i}$ *is an integer, which represents the index of the current block in the chain. It is equal to the number of the ancestor blocks. The genesis block has number 0.*
- ***state_root:*** *formally indicated as* $\boldsymbol{H_r}$ *is the root of the Merkle trie, whose leaves implement the storage for the system.*
- ***extrinsics_root:*** *is the root of the Merkle trie, whose leaves represent individual extrinsic being validated in this block. This element is formally referred to as* $\boldsymbol{H_e}$.
- ***digest:*** *this field is used to store any chain-specific auxiliary data, which could help the light clients interact with the block without the need of accessing the full storage. Polkadot RE does not impose any limitation or specification for this field. It essentially can be a byte array of any length. This field is indicated as* $\boldsymbol{H_d}$

2.2 Justified Block Header

The Justified Block Header is provided by the consensus engine and presented to the Polkadot RE, in order for the block to be appended to the blockchain. It contains the following parts:

- **block_header** the complete block header as defined in Section 2.1 and denoted by $\text{Head}(B)$.
- **justification**: as defined by the consensus specification denoted by $\text{Just}(B)$ [link this to its definition from consensus].
- **authority Ids**: This is the list of the Ids of authorities, which have voted for the block to be stored and is formally referred to as $A(B)$. An authority Id is 32bit.

2.3 Extrinsics

Each block also contains a list of extrinsics. Polkadot RE does not specify or limit the internal of each extrinsics beside the fact that each extrinsics is a blob of encoded data. The extrinsics_root should correspond to the root of the Merkle trie, whose leaves are made of the block extrinsics list.

3 Entry into Runtime

4 API

4.1 Block Submission and Validation

Block validation is the process, by which the client asserts that a block is fit to be added to the blockchain. That is to say, the block is consistent with the world state and transitions from the state of the system to a new valid state.

Blocks can be handed to the Polkadot RE both from the network stack and from consensus engine.

Both the runtime and the Polkadot RE need to work together to assure block validity. This can be accomplished by Polkadot RE invoking `execute_block` entry into the runtime as a part of the validation process.

Polkadot RE implements the following procedure to assure the validity of the block:

Algorithm 1

IMPORT-AND-VALIDATE-BLOCK(B, Just(B))
1 VERIFY-BLOCK-JUSTIFICATION(B, Just(B))
2 Verify $H_{p(B)} \in$ Blockchain.
3 State-Changes = Runtime.EXECUTE-BLOCK(B)
4 UPDATE-WORLD-STATE(State-Changes)

4.2 Storage Access

5 State Storage and the Storage Trie

For storing the state of the system, Polkadot RE implements a hash table storage where the keys are used to access each data entry state. There is no limitation on neither the size of the key nor the size of the data stored under them, beside the fact that thay are byte arrays.

To authenticate the state of the system, the stored data is re-arranged and hashed in a *radix 16 tree* also known as *base-16 modified Merkle Patricia Tree*, which hereafter we simply refer to as the ***Trie***, in order to compute the hash of the whole state storage consistently and efficiently at any given time.

Also modification has been done in storing the nodes' hash in the Merkle tree structure to save space on entries storing small entries.

Because the Tri is used to compute the *state root*, H_r, (see Definition 7), which is used to authenticate the validity of the state database, Polkadot RE follows a rigorous encoding algorithm to compute the values stored in the trie nodes to ensure that the computed Merkle hash, H_r, matches across clients.

5.1 The General Tree Structure

As the trie is a radix 16 tree, in this sense, each key value identifies a unique node in the tree. However, a node in a tree might or might not be associated with a key in the storage.

To identify the node corresponding to a key value, k, first we need to encode k in a uniform way:

Definition 8. *The for the purpose labeling the branches of the Trie key k is encoded to k_{enc} using KeyEncode functions:*

$$k_{\text{enc}} := (k_{\text{enc}_1}, \ldots, k_{\text{enc}_{2n}}) := \text{KeyEncode}(k) \tag{1}$$

such that:

$$\text{KeyEncode}(k): \begin{cases} \mathbb{B} & \rightarrow \text{Nibbles}_4 \\ k := (b_1, \ldots, b_n) := & \mapsto (b_1^1, b_1^2, b_2^1, b_2^2, \ldots, b_n^1, b_n^2) \\ & := (k_{\text{enc}_1}, \ldots, k_{\text{enc}_{2n}}) \end{cases}$$

where Nibble_4 *is the set of all nibbles of 4-bit arrays and* b_i^1 *and* b_i^2 *are 4-bit nibbles which are the the little endian representation of* b_i*:*

$$(b_i^1, b_i^2) := (b_1 \text{mod}\, 16, b_2 / 16)$$

where mod is the remainder and / is the integer division operators.

By looking at k_{enc} as a sequence of nibbles, can walk the radix tree to reach the node identifying the storage value of k.

5.2 The Merkle proof

To prove the consistency of the state storage across the network and its modifications efficiently, the Merkle hash of the storage trie needs to be computed rigorously.

The Merkle hash of the trie is computed recursively. As such, hash value of each node depends on the hash value of all its children and also on its value. Therefore, it suffices to define how to compute the hash value of a typical node as a function of the hash value of its children and its own value.

Definition 9. *Suppose node N of storage state trie has key value* k_N*, and parent key value of* $k_{P(N)}$*, such that:*

$$\text{KeyEncode}(k_N) = (k_{\text{enc}_1}, \ldots, k_{\text{enc}_{i-1}}, k_{\text{enc}_i}, \ldots, k_{\text{enc}_{2n}})$$

and

$$\text{KeyEncode}(k_{P(N)}) = (k_{\text{enc}_1}, \ldots, k_{\text{enc}_{i-1}})$$

We define

$$\text{pk}_N := (k_{\text{enc}_i}, \ldots, k_{\text{enc}_{2n}})$$

to be the ***the partial key*** *of N.*

Definition 10. *For a trie node N, **Node Prefix** function is a value specifying the node type as follows:*

$$\text{NodePrefix}(N) := \begin{cases} 1 & N \textit{is a leaf node} \\ 254 & N \textit{is a branch node without value} \\ 255 & N \textit{is a branch node with value} \end{cases}$$

Definition 11. *For a given node N, with partial key of pk_N and Value v, the **encoded representation** of N, formally referred to as $\text{Enc}_{\text{Node}}(N)$ is determined as follows, in case which:*

- *N is a leaf node:*

$$\text{Enc}_{\text{Node}}(N) := \text{Enc}_{\text{len}}(N) || \text{HPE}(\text{pk}_N) || \text{Enc}_{\text{SC}}(v)$$

- *N is a branch node:*

$$\begin{aligned}&\mathit{Enc}_{\mathit{Node}}(\mathit{N}) := \\ &\text{NodePrefix}(N) || \text{ChildrenBitmap}(N) \| \text{HPE}_{\text{PC}}(v) || \text{Enc}_{\text{SC}}(\text{Enc}_{\text{Node}}) || \\ &\text{Enc}_{\text{SC}}(N_{C_1}) \ldots \text{Enc}_{\text{SC}}(N_{C_n})\end{aligned}$$

Where $N_{C_1} \ldots N_{C_n}$ with $n \leqslant 16$ are the children nodes of N.

Definition 12. *For a given node N, the **Merkle value** of N, denoted by $H(N)$ is defined as follows:*

$$\begin{aligned}&H: \mathbb{B} \rightarrow \bigcup_{i=0}^{32} \mathbb{B}_i \\ &H(N): \begin{cases} \text{Enc}_{\text{Node}}(N) & \|\text{Enc}_{\text{Node}}(N)\| < 32 \\ \text{Hash}(\text{Enc}_{\text{Node}}(N)) & \|\text{Enc}_{\text{Node}}(N)\| \geqslant 32 \end{cases}\end{aligned}$$

6 Extrinsics trie

Although the same radix-16 tree is being used to verify the consistency of the Extrinsics data, unlike the Storage trie, Polkadot RE *does not* use a Merkle tree structure to accomplish this task. Instead, it stores the entire trie in a byte array structure and finally computes its hash as a byte array, as defined in Definition 13.

Definition 13. ***Closed Form Trie*** *[TODO: Define according to substerate-submods/trie/trie-root/src/lib.rs]*

7 Auxilary Encodings

7.1 Simple Codec

Polkadot RE uses *simple codec* to encode byte arrays to provide canonical encoding and to produce consistent hash values across their implementation, including the Merkle hash proof for the State Storage.

Definition 14. *The* ***simple codec*** *of a byte array A:*

$$A := b_1\, b_2 \ldots b_n$$

such that $n < 2^{536}$ is a byte array refered to $\mathrm{Enc}_{\mathrm{SC}}(A)$ and defined as follows:

$$\mathrm{Enc}_{\mathrm{SC}}(A) := \begin{cases} l_1\, b_1\, b_2 \ldots b_n & 0 \leqslant n < 2^6 \\ i_1\, i_2\, b_1 \ldots b_n & 2^6 \leqslant n < 2^{14} \\ j_1\, j_2\, j_3\, b_1 \ldots b_n & 2^{14} \leqslant n < 2^{30} \\ k_1\, k_2 \ldots k_m\, b_1 \ldots b_n & 2^{14} \leqslant n \end{cases}$$

in which:

$l_1^1\, l_1^0 = 00$
$i_1^1\, i_1^0 = 01$
$j_1^1\, j_1^0 = 10$
$k_1^1\, k_1^0 = 11$

and n is stored in $\mathrm{Enc}_{\mathrm{SC}}(A)$ in little-endian format in base-2 as follows:

$$n = \begin{cases} l_1^7 \ldots l_1^3\, l_1^2 & n < 2^6 \\ i_2^7 \ldots i_2^0\, i_1^7 \ldots i_1^2 & 2^6 \leqslant n < 2^{14} \\ j_4^7 \ldots j_4^0\, j_3^7 \ldots j_1^7 \ldots j_1^2 & 2^{14} \leqslant n < 2^{30} \\ k_2 + k_3\, 2^8 + k_4\, 2^{2\cdot 8} + \cdots + k_m\, 2^{(m-2)8} & 2^{30} \leqslant n \end{cases}$$

where:

$$m = l_1^7 \ldots l_1^3\, l_1^2 + 4$$

7.2 Hex Encoding

Practically it is more convenient and efficient to store and process data which is stored in a byte array. On the other hand, radix-16 tree keys are broken in 4-bits nibbles. Accordingly, we need a method to encode sequences of 4-bits nibbles into byte arrays canonically:

Definition 15. *Suppose that* $\mathrm{PK} = (k_1, \ldots, k_n)$ *is a sequence of nibbles, then*

$$\mathrm{Enc}_{\mathrm{HE}}(\mathrm{PK}) := \begin{cases} \mathrm{Nibbles}_4 & \rightarrow \mathbb{B} \\ \mathrm{PK} = (k_1, \ldots, k_n) & \mapsto \begin{cases} (0, k_1 + 16\, k_2, \ldots, k_{2i-1} + 16\, k_{2i}) & n = 2\, i \\ (k_1, k_2 + 16\, k_3, \ldots, k_{2i} + 16\, k_{2i+1}) & n = 2\, i + 1 \end{cases} \end{cases}$$

7.3 Partial Key Encoding

Definition 16. *Let N be a node in the storage state trie with Partial Key* PK_N*. We define the* ***Partial key length encoding*** *function, formally referred to as* $\mathrm{Enc}_{\mathrm{len}}(N)$ *as follows:*

$$\begin{array}{l} \mathrm{Enc}_{\mathrm{len}}(N) & := \\ \mathrm{NodePrefix}(N) & + \\ \begin{cases} (\|\mathrm{PK}_N\|) & \mathrm{NisleafNode} \quad \& \quad \|\mathrm{PK}_N\| < 127 \\ (127) || (\mathrm{LE}(\|\mathrm{PK}_N\| - 127)) & \mathrm{NisaleafNode} \quad \& \quad \|\mathrm{PK}_N\| \geqslant 127 \end{cases} \end{array}$$

where NodePrefix *function is defined in Definition 10.*