Speculo-Bridge (Generic Messaging)

Technical Implementation

Abstract

> This specification assumes that the underlying transfer protocol already supports bidirectional arbitrary message transmission, either through full consensus-level verification or light client-based relayers. For bridges that only support token transfers, please refer to the Speculo-Bridge (Token Transfer Only) specification.
>
> All messages should be wrapped in generic JSON format.
>
> > This JSON payload should always be decodable by an arbitrary set of blockchains and bridges - i.e. data transfer shouldn't be affected by which type of bridging protocol is being used.
>
> The bridge specification shall support the relaying of **four** message types: `initWithdrawReq`, `initWithdrawRes`, `finishWithdrawReq`, and `finishWithdrawRes`.
>
> > This message structure is based on a request - `ACK` architecture, but simplified - as we assume that transport reliability is guaranteed at the communication layer.

Architecture

> Data serialization & deserialization functionality should be processed at the underlying bridging protocol - this spec only defines messages being transferred on the upper JSON layer.
>
> Anchor only requires *one* functionality over the bridge: **withdrawing staking rewards** accumulated on a remote chain where bAssets were first issued.
>
> The full architecture are as follows:
>
> `initWithdrawReq`: sent by the Anchor system to the remote chain. Initializes a `Withdraw` request.
>
> `initWithdrawRes`: sent by the remote chain to the Anchor system. Either confirms execution of `initWithdrawRes`, or returns an error.
>
> `finishWithdrawReq`: sent by the Anchor system to the remote chain. Closes an asynchronous `Withdraw` request.
>
> `finishWithdrawRes`: sent by the remote chain to the Anchor system. Finalizes execution, and either returns claimed staking rewards, or returns an error.

Messages & Implementation

Note that this specification does not define anything on the underlying transport layer: transport architectures should be defined by the bridge vendor.

Assuming either a packet-based, generic messaging architecture or a light-client based state syncing solution, the bridge system should be able to relay the following messages, formatted in JSON-RPC (https://www.jsonrpc.org/specification):

`initWithdrawReq`

This message should be in the following format:

```
{
“jsonrpc”: “2.0”,
“method”: “initWithdraw”,
“params”: {
          “denom”: <string>,
```

```
          “owningAccount”: <bytes>,
          “anchorAccount”: <bytes>,
          “amount”: <uint64>,
          “requestSignature”: <bytes>
          },
“id”: <uint16>
}
```

with the parameters being:

`denom`: the token denominator being requested for reward withdrawal - e.g. `ATOM`, `DOT`, `SOL`

`owningAccount`: the address on the remote chain owning the original bAssets before it was wrapped and sent over to the Anchor system, in the remote chain’s address format - e.g. `terra123...`, `cosmos123...`, `1FRMM8...`

`anchorAccount`: the address holding wbAssets and requesting rewards on the remote chain, in the Anchor system’s address format - e.g. `terra123...`

`amount`: the amount of bAssets that we are requesting rewards on, and sending over to the remote chain.

`requestSignature`: signed payload of a withdrawal request JSON, assuming that the `requestSignature` field itself is set as `null`. The signature should be generated from the `anchorAccount`’s key.

Other values should follow JSON-RPC specifications.

`initWithdrawRes`

This message should be in the following format:

```
{
“jsonrpc”: “2.0”,
“result”: {
          “owningAccount”: <bytes>,
          “requestHash”: <bytes>,
          "responseSignature": <bytes>
          },
“id”: <uint16>
}
```

with the parameters being:

`owningAccount`: the address on the remote chain approving this withdrawal request, in the remote chain’s address format - e.g. `cosmos123...`, `1FRMM8...`

`requestHash`: a `SHA-256` hash value of the `initWithdrawReq` object.

`responseSignature`: signed payload of a withdrawal request JSON, assuming that the `responseSignature` field itself is set as `null`. The signature should be generated from the `owningAccount`’s key.

Other values should follow JSON-RPC specifications.

or, in the case of an error:

```
{
“jsonrpc”: “2.0”,
“error”: {
          “code”: <int16>,
          “message”: <string>
```

```
        },
“id”: <uint16>
}
```

with the parameters being:

> `code`: RPC standard response error code, as defined with http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php. Speculo specific error code definitions shall be defined with a separate document.
>
> `message`: a short description of the cause of this error.
>
> Other values should follow JSON-RPC specifications.

`finishWithdrawReq`

This message should be in the following format:

```
{
“jsonrpc”: “2.0”,
“method: ”finishWithdraw“,
“params”: {
        “initResponseSignature”: <bytes>,
        “responseHash”: <bytes>,
        “anchorAccount”: <bytes>,
        “requestSignature”: <bytes>
        },
“id”: <uint16>
}
```

with the parameters being:

> `initResponseSignature`: the `responseSignature` value under the `initWithdrawRes` response object. This is included as an acknowledgement of the remote chain’s signature.
>
> `responseHash`: a `SHA-256` hash value of the `initWithdrawRes` object.
>
> `anchorAccount`: the address holding wbAssets and requesting rewards on the remote chain, in the Anchor system’s address format - e.g. `terra123...`
>
> `requestSignature`: signed payload of a finishWithdrawal request JSON, assuming that the `responseSignature` field itself is set as `null`. The signature should be generated from the `anchorAccount`’s key.
>
> Other values should follow JSON-RPC specifications.

`finishWithdrawRes`

This message should be in the following format:

```
{
“jsonrpc”: “2.0”,
“result”: {
        “owningAccount”: <bytes>,
        “bAssetAmount”: <uint64>,
```

```
            “rewardAmount”: <uint64>,
            “requestHash”: <bytes>,
            "responseSignature": <bytes>
            },
“id”: <uint16>
}
```

with the parameters being:

- `owningAccount`: the address on the remote chain approving this withdrawal request, in the remote chain's address format - e.g. `cosmos123...`, `1FRMM8...`
- `bAssetAmount`: the amount of bAssets initially requested to withdraw rewards from.
- `rewardAmount`: the amount of vanilla Assets claimed as staking rewards. This should be calculated as: `totalStakedTokens`* `bAssetAmount` / `bAssetSupply` .
- `requestHash`: a `SHA-256` hash value of the `finishWithdrawReq` object.
- `responseSignature`: signed payload of a withdrawal response JSON, assuming that the `responseSignature` field itself is set as `null`. The signature should be generated from the `owningAccount`'s key.
- Other values should follow JSON-RPC specifications.

or, in the case of an error:

```
{
“jsonrpc”: “2.0”,
“error”: {
            “code”: <int16>,
            “message”: <string>
            },
“id”: <uint16>
}
```

with the parameters being:

- `code`: RPC standard response error code, as defined with http://xmlrpc-epi.sourceforge.net/specs/rfc.fault_codes.php. Speculo specific error code definitions shall be defined with a separate document.
- `message`: a short description of the cause of this error.
- Other values should follow JSON-RPC specifications.