Overview

This document specifies the Interchain Transactions module for the Neutron network.

The Interchain Transactions module manages the creation of IBC Accounts and executing interchain transactions on behalf of CosmWasm smart contracts. The current implementation allows a smart contract to:

1. Register multiple interchain accounts on a remote zone using an existing IBC connection;
2. Execute transactions with multiple messages on a remote zone;
3. Process theOnChanOpenAck
4. ,Acknowledgement
5. andTimeout
6. events as they are delivered by a relayer.

IBC events

Registering an interchain account or executing an interchain transaction are asynchronous actions. In most cases, the respective handlers of the Interchain Transactions module immediately return an empty successful response. The "real" response (with information about the status of execution on a remote zone) is later delivered in a separate IBC packet by a relayer. We call such packets theIBC events .

A smart contract that tries to register an interchain account or to execute an interchain transaction naturally expects to receive the IBC events related to these actions. The Interchain Transactions module solves this task by passing these IBC events to the smart contract using aSudo() call and a custommessage scheme . You can find a complete list of IBC events for each module message in themessages section.

Sudo errors handling

Interchaintxs module configured the following way, all the errors from a sudo handler are being suppressed by acontract manager middleware , sudo handler is limited withLIMIT amount of gas

Importing interchaintxs module

If you use interchaintxs module in your application and if your Sudo handler fails, the acknowledgment will be marked as processed inside the IBC module anyway, without any notes about an error saved into the store. This will make the IBC relayers to successfully submit the acknowledgment and get the fees.

Note You can use acontracts manager SudoLimitWrapper as a wrapper for SudoKeeper, exactly like neutronconfigured the module

Note We strongly recommend developers to write Sudo handlers very carefully and keep them as simple as possible. If you do want to have elaborate logic in your handler, you should verify the acknowledgement data before making any state changes; that way you can, if the data received with the acknowledgement is incompatible with executing the handler logic normally, return anOk() response immediately, which will prevent the acknowledgement from being resubmitted.

Note : there is no dedicated event for a closed channel (ICA disables all messages related to closing the channels) . Your channel, however, can still be closed if a packet timeout occurs; thus, if you are notified about a packet timeout, you can be sure that the affected channel was closed. Please note that it is generally a good practice to set the packet timeout for your interchain transactions to a really large value.

If the timeout occurs anyway, you can just executeRegisterInterchainAccount message again to recover access to your interchain account.Note Keep in mind, new channel is created

Sudo Handlers

The interchaintxs module in neutrond configured the following way.

Wasmd Sudo handler wrapped withSudoLimitWrapper

And acknoledgement packet follows the wayOnAcknowledgementPacket -->SudoLimitWrapper -->Wasmd Sudo handler

UsingSudoLimitWrapper has two purposes:

1. Suppress the sudo handler error itself, and mark the ibc acknowledgement packet as received and processed. Other way, the error makes relayer send an acknowledgement again and again. Information about an unsuccessfully processed ack is stored instate
2. .

3. Limit the amount of gas available for sudo handler execution. Out of gas panic will later be captured bySudoLimitWrapper
4. and converted into an error.

Failed interchain txs

Not every interchaintx executes succesfully on a remote network. Some of them fail to execute with errors and then you get ibc acknowledgement withError type. The error is passed into the caller contract via sudo call with SudoMsg::Errorvariant

Unfortunately, to avoid the nondeterminism associated with error text generation, the error text is severely truncated byredact down to the error code without any additional details, before converting into AcknowledgementError.

Find the error text is possible if host chain includes ibc-go v7.2.3+, v7.3.2+, v8.0.1+, v8.1+ which include patch5541 q interchain-accounts host packet-events

Where:

- binary
- is a binary on the chain you are working with (the remote chain)
- seq-id
-
    - sequence ID of the IBC message sent to the remote chain. The seq-id is returned to the contract in the SubmitTx
- response
- channel-id
- is the ID of the ICA's channel on the remote chain's side. You should know it from registrationprocedure
- viaSudoMsg::OpenAck
- fromcounterparty_channel_id
- field. If you missed it you can always get counterparty channel-id with CLI commandneutrond q ibc channel end
- src-channel-id
- is the channel you intechain account associated with.
- src-port
- is the port you interchain account is associated with.
- You should know bothsrc-channel-id
- andsrc-port
- from registrationprocedure
- . Alsosrc-port
- isicacontroller-.
- whereica_id
- defined by you during ica registration.

Output example (filtered events):

{ "type" :

"ibccallbackerror-ics27_packet" , "attributes" :

[{ "key" :

"ibccallbackerror-module" , "value" :

"interchainaccounts" , "index" :

true } , { "key" :

"ibccallbackerror-host_channel_id" , "value" :

"channel-2" , "index" :

true } , { "key" :

"ibccallbackerror-success" , "value" :

"false" , "index" :

true } , { "key" :

"ibccallbackerror-error" , "value" :

"invalid validator address: decoding bech32 failed: invalid separator index -1: invalid address" , "index" :

true }] } On earlier versions of ibc-go it's barely possible to get full text error due topatch .

In the IBC error acknowledgement you get ABCI error and a code, e.g.codespace: wasm, code: 5 where codespace usually isModuleName , andcode is uniq code for the module.codespace andcode pair uniq for the whole app. You can find the error description in source code. Usualy all the error of the module areplaced inx//types/errors.go wheremodule is the module where the error was thrown

Relaying

Neutron introduces smart-contract level callbacks for IBC packets. From an IBC relayer's perspective, this means that custom application logic can be executed when a packet is submitted to Neutron, which can potentially drain the relayer's funds. This naturally brings us to a situation in which protocols would prefer to set up their own relayers and restrict the channels they are willing to relay for. For example, inHermes you can do this by adding achains.packet_filter config:

[chains.packet_filter] policy

=

'allow' list

=

[

allow relaying only for chanels created by a certain contract

['icacontroller-neutron14hj2tavq8fpesdwxxcu44rty3hh90vhujrvcmstl4zr3txmfvw9s5c2epq*' ,

'*'] ,] Note: you can have a look at theMsgRegisterInterchainQuery documentation in theMessages chapter to learn how IBC port naming works. Please refer to theIBC Relaying section for full IBC relaying documentation. Previous Messages Next Messages