Overview

Motivation

The motivation of IBC-rate-limit comes from the empirical observations of blockchain bridge hacks that a rate limit would have massively reduced the stolen amount of assets in:

- Polynetwork Bridge Hack (611 million)
- BNB Bridge Hack (586 million)
- Wormhole Bridge Hack (326 million)
- Nomad Bridge Hack (190 million)
- Harmony Bridge Hack (100 million)
-
    - (Would require rate limit + monitoring)
- Dragonberry IBC bug
- (can't yet disclose amount at risk, but was saved due to being found first by altruistic Osmosis core developers)

In the presence of a software bug on Neutron, IBC itself, or on a counterparty chain, we would like to prevent the bridge from being fully depegged. This stems from the idea that a 30% asset depeg is ~infinitely better than a 100% depeg. Itscrazy that today these complex bridged assets can instantly go to 0 in event of bug. The goal of a rate limit is to raise an alert that something has potentially gone wrong, allowing validators and developers to have time to analyze, react, and protect larger portions of user funds.

The thesis of this is that, it is worthwhile to sacrifice liveness in the case of legitimate demand to send extreme amounts of funds, to prevent the terrible long-tail full fund risks. Rate limits aren't the end-all of safety controls, they're merely the simplest automated one. More should be explored and added onto IBC!

Rate limit types

We express rate limits in time-based periods. This means, we set rate limits for (say) 6-hour, daily, and weekly intervals. The rate limit for a given time period stores the relevant amount of assets at the start of the rate limit. Rate limits are then defined on percentage terms of the asset. The time windows for rate limits are currentlynot rolling, they have discrete start/end times.

We allow setting separate rate limits for the inflow and outflow of assets. We do all of our rate limits based on thenet flow of assets on a channel pair. This prevents DOS issues, of someone repeatedly sending assets back and forth, to trigger rate limits and break liveness.

We currently envision creating two kinds of rate limits:

- Per denomination rate limits* allows safety statements like "Only 30% of Stars on Neutron can flow out in one day" or "The amount of Atom on Neutron can at most double per day".
- Per channel rate limits* Limit the total inflow and outflow on a given IBC channel, based on "USDC" equivalent, using Neutron as the price oracle.

We currently only implement per denomination rate limits for non-native assets. We do not yet implement channel based rate limits.

Currently these rate limits automatically "expire" at the end of the quota duration.

Instantiating rate limits

Today all rate limit quotas must be set manually by governance. In the future, we should design towards some conservative rate limit to add as a safety-backstop automatically for channels. Ideas for how this could look:

- One month after a channel has been created, automatically add in some USDC-based rate limit
- One month after governance incentivizes an asset, add on a per-denomination rate limit.

Definitely needs far more ideation and iteration!

Parameterizing the rate limit

One element is we don't want any rate limit timespan that's too short, e.g. not enough time for humans to react to. So we wouldn't want a 1 hour rate limit, unless we think that if its hit, it could be assessed within an hour.

Handling rate limit boundaries

We want to be safe against the case where say we have a daily rate limit ending at a given time, and an adversary attempts to attack near the boundary window. We would not like them to be able to "double extract funds" by timing their extraction near a window boundary.

Admittedly, not a lot of thought has been put into how to deal with this well. Right now we envision simply handling this by saying if you want a quota of duration D, instead include two quotas of duration D, but offset byD/2 from each other.

Ideally we can change windows to be more 'rolling' in the future, to avoid this overhead and more cleanly handle the problem. (Perhaps rolling ~1 hour at a time)

Inflow parameterization

The "Inflow" side of a rate limit is essentially protection against unforeseen bug on a counterparty chain. This can be quite conservative (e.g. bridged amount doubling in one week).

It does get more complex when the counterparty chain is itself a DEX, but this is still much more protection than nothing.

Outflow parameterization

The "Outflow" side of a rate limit is protection against a bug on Neutron OR IBC. This has potential for much more user-frustrating issues, if set too low. E.g. if there's some event that causes many people to suddenly withdraw many STARS or many USDC.

So this parameterization has to contend with being a tradeoff of withdrawal liveness in high volatility periods vs being a crucial safety rail, in event of on-Neutron bug.

Cosmwasm Contract Concepts

Note: the contract implementation can be foundhere Something to keep in mind with all of the code, is that we have to reason separately about every item in the following matrix:

Native Token Non-Native Token Send Native Token Send Non-Native Token Receive Native Token Receive Non-Native Token Timeout Native Send Timeout Non-native Send (Error ACK can reuse the same code as timeout)

The tracking contract uses the following concepts

1. RateLimit
2.
    - tracks the value flow transferred and the quota for a path.
3. Path
4.
    - is a (denom, channel) pair.
5. Flow
6.
    - tracks the value that has moved through a path during the current time window.
7. Quota
8.
    - is the percentage of the denom's total value that can be transferred through the path in a given period of time (duration)

Messages

The contract specifies the following messages:

Query

- GetQuotas - Returns the quotas for a path

Exec

- AddPath - Adds a list of quotas for a path
- RemovePath - Removes a path
- ResetPathQuota - If a rate limit has been reached, the contract's governance address can reset the quota so that transfers are allowed again

Sudo

Sudo messages can only be executed by the chain.

- SendPacket - Increments the amount used out of the send quota and checks that the send is allowed. If it isn't, it will return a RateLimitExceeded error
- RecvPacket - Increments the amount used out of the receive quota and checks that the receive is allowed. If it isn't, it will return a RateLimitExceeded error
- UndoSend - If a send has failed, the undo message is used to remove its cost from the send quota

All of these messages receive the packet from the chain and extract the necessary information to process the packet and determine if it should be the rate limited.

Necessary information

To determine if a packet should be rate limited, we need:

- Channel: The channel on the Neutron side:packet.SourceChannel
- for sends, andpacket.DestinationChannel
- for receives.
- Denom: The denom of the token being transferred as known on the Neutron side (more on that below)
- Channel Value: The total value of the channel denominated inDenom
- (i.e.: channel-17 is worth 10k osmo).
- Funds: the amount being transferred

Notes on Channel

The contract also supports quotas on a custom channel called "any" that is checked on every transfer. If either the transfer channel or the "any" channel have a quota that has been filled, the transaction will be rate limited.

Notes on Denom

We always use the the denom as represented on Neutron. For native assets that is the local denom, and for non-native assets it's the "ibc" prefix and the sha256 hash of the denom trace (ibc/...).

Sends

For native denoms, we can just use the denom in the packet. If the denom is invalid, it will fail somewhere else along the chain. Example result:uosmo

For non-native denoms, the contract needs to hash the denom trace and append it to theibc/ prefix. The contract always receives the parsed denom (i.e.:transfer/channel-32/uatom instead ofibc/27394FB092D2ECCD56123C74F36E4C1F926001CEADA9CA97EA622B25F41E5EB2). This is because of the order in which the middleware is called. When sending a non-native denom, the packet containstransfer/source-channel/denom as it is built on therelay.SendTransfer() in the transfer module and then passed to the middleware. Example result:ibc/

Receives

This behaves slightly different if the asset is an Neutron asset that was sent to the counterparty and is being returned to the chain, or if the asset is being received by the chain and originates on the counterparty. In ibc this is called being a "source" or a "sink" respectively.

If the chain is a sink for the denom, we build the local denom by prefixing the port and the channel (transfer/local-channel) and hashing that denom. Example result:ibc/

If the chain is the source for the denom, there are two possibilities:

- The token is a native token, in which case we just remove the prefix added by the counterparty. Example result:uosmo
- The token is a non-native token, in which case we remove the extra prefix and hash it. Example resultibc/

Notes on Channel Value

We have iterated on different strategies for calculating the channel value. Our preferred strategy is the following:

- For non-native tokens (ibc/...
-), the channel value should be the supply of those tokens in Neutron
- For native tokens, the channel value should be the total amount of tokens in escrow across all ibc channels

The later ensures the limits are lower and represent the amount of native tokens that exist outside Neutron. This is beneficial as we assume the majority of native tokens exist on the native chain and the amount "normal" ibc transfers is proportional to the tokens that have left the chain.

This strategy cannot be implemented at the moment because IBC does not track the amount of tokens in escrow across all channels (github issue). Instead, we use the current supply on Neutron for all denoms (i.e.: treat native and non-native tokens the same way). Once that ticket is fixed, we will update this strategy.

Caching

The channel value varies constantly. To have better predictability, and avoid issues of the value growing if there is a potential infinite mint bug, we cache the channel value at the beginning of the period for every quota.

This means that if we have a daily quota of 1% of the osmo supply, and the channel value is 1M osmo at the beginning of the quota, no more than 100k osmo can transferred during that day. If 10M osmo were to be minted or IBC'd in during that period, the quota will not increase until the period expired. Then it will be 1% of the new channel value (~11M)

Integration

The rate limit middleware wraps thetransferIBCModule and is added as the entry route for IBC transfers.

The module is also provided to the underlyingtransferIBCModule as itsICS4Wrapper ; previously, this would have pointed to a channel, which also implements theICS4Wrapper interface.

This integration can be seen inneutron/app/app.go Previous Metrics Next Overview