Celestia's primary users are rollups who want to post blobs. Now, different projects will have different requirements. Celestia can move in a direction that gives those users better UX when posting blobs. You might think posting blobs is a one-dimensional problem with little room to customize. Still, various ways exist to improve rollup infrastructure built on Celestia by enabling customizability. The post will touch on rollup resource pricing, the woods attack, intra-namespace priority, account abstraction, and leveraging zero-knowledge proofs for transaction preconditions.

Proplem Space

This section will show various ways people would want to customize their namespace or how they interact with their namespace.

Limits on the namespace & Resource Pricing

As Celestia's block size increases, the permissionless nature of posting a blob to a namespace could cause an exhaustion attack on the rollups stack. I am here referring to the Woods attack.

This document describes a denial-of-service (DoS) attack that targets rollup nodes trying to sync with Celestia's latest rollup state. While bootstrapping, these nodes download the headers of all past rollup blocks to determine the canonical tip of the rollup chain. Thus, the time required to complete bootstrapping depends on the number of rollup blocks previously published on the parent chain. In this context, attackers could considerably delay bootstrapping by publishing many invalid rollup blocks on the parent chain and forcing syncing nodes to download all their headers.

Proposed solutions range from limiting the size of a namespace to making the cost of filling a namespace quadratic, or making it dependent on previous blocks fullness. Another approach could be pricing each namespace using an EIP-1559 style mechanism.

Other limits rollups may desire on a namespace include the size of bytes that fit into it, the number of blobs, or upper and lower limits on the size of the blobs. Another limit could be that every blob in a namespace has to be unique. If a DA-Layer could provide these limits, the rollups built on top would benefit.

All the factors could influence upstream resource pricing of negative externalities.

Such negative externalities could be

- the download amount of raw bytes
- the decompression and decoding of blobs
- various semantics checks of validity per blob
- zk proving that spam data is spam data if the proof has to go over the whole namespace (completeness)

Population of additional information in the Blob Shares

Some Rollup SDKs and libraries want to be able to work out the signer/author of the PFB as part of validating a blob. For example, this is the case for sovreign - sdk as they verify the signers of the PFB, which requires them to download the PFB namespaces. One of the solutions provided in "PFB signer namespace" is to populate the first share with the signer who posted the blob.

Other metadata that the signer would want to know could be how much the user paid in gas or tip to post to Celestia. With these types of metadata, you can customize your namespace and do not need to read the PFB-Namespace for pfb-submission-related data.

Conditions on submission

Users might want to limit the namespace to other conditions. For example, the simplest condition could be that only certain signers can post to a namespace or more complex conditions that work well with the modular stack built on top that we will explore later.

Solution Space

This section will show various ways how Celestia could solve the problems listed above and enable a unique blob submission experience.

More transaction types

The first intuition is adding new transaction types to the state machine. Each condition could get its transaction type and its reserved namespace. Of course, you could give the user options to specify while submitting the blob.

Another option could be to use feegrant and authz-like modules. These already offer submission optionality in fee usage and could be extended.

Namespace encoding: namespace prefix = condition

A namepsace is 29 bytes long and contains the version and id.

Currently, the first 19 bytes are 0 in every namespace except some reserved namespaces at the end of the spectrum. The first byte is 0 as we only have 1 version so far, and the next 18 bytes are 0, as well as a consensus rule for supported user-specifiable namespaces. We can use parts of the 18 0-byte prefix to indicate what rules apply in the namespace. There would be a deterministic mapping of bytes to rules.

Example:

If you want to limit the size of the namespace, flip bit #x

followed by the number of shares you want it limited to.

10000

could be read as limiting the number of shares to 1 share.

10001

is a two-share limit, 10010

is a four-share limit, 10011

is an eight-share limit, and so on. If it starts with 0, you can use the space for other rules. You can get creative and count the number of 0s. I leave it as an exercise for the reader to imagine what an efficient encoding might look like.

Assuming there exists a predefined mapping of bytes to a rule set, users can submit transactions with the API they are used to. Just by specifying a particular namespace, they can get the desired rule set. We would retain the ability to submit permissionlessly and have 10 bytes worth of namespaces per rule set permutation if we use up all 18 bytes for rules.

Celestia script: script hash = condition

If you want a more thorough ruleset, Celestia could add a scripting language like Bitcoin'sScript. Funny enough, Bitcoin's scripting makes it more expressive than Celestia today, with use cases like the BitVM making Celestia even more minimalistic than Bitcoin. We could change that to introduce specific rules for blob submission.

While executing the PFB to a particular namespace, a script would be included in the transaction. If the script's hash equals the namespace destination, you can post it to that namespace. Additionally, the validator will look at the script and enforce the rules before you submit it to the namespace.

This will give us much more flexibility in what kind of rules we can post. It is also future-proof, as we can always add or remove OP codes.

The collision-resistance of 2 scripts might be a problem. If we only take the first 18 bytes of the hash, we will not get the full collision resistance. If we want to encode the hash in the namespace, we will either forfeit the full collision resistance or not keep the permissionless nature of posting to a namespace and a many-to-1 relationship between namespaces and a ruleset. Going with 28 bytes for the highest collision resistance would no longer be backward compatible, and every submission would require a script.

We could also increase the namespace ID from 28 to 42 and bump the namespace version by 1. Here, the first 32 bytes would be the hash for the script, and the next 10 bytes would be the ID of how users are used to it. A change in the size of the namespace would be a significant breaking change to NMT data structure and could make it not backward compatible. The reason against doing that is that hashing the NMT would be more expensive. Currently, the namespace is only 29 bytes and not 32, creating three more hashing operations per inner node in the NMT. The number of hash operations influences the time it takes to generate zk-proofs of inclusion of inner nodes under the data root.

If we are ok with the hash not being encoded in the namespace as other rules, we can relax the solution space as we only need to add the metadata somewhere in the square. The final and most straightforward solution is to add the hash as a suffix to the namespace in the first share of the blob. This way, we don't have to change the namespace encoding. We could flip a particular bit in the namespace to indicate that a script hash was used and append that hash to the namespace in the first share.

In general, rollups would only follow the forkchoice rule of the script hash that they had agreed upon beforehand, ignoring other blobs when traversing a namespace.

Celestia snark account: verification key = condition

In the current specification, theSNARK-Accounts of Snacc have two functionallities, withdrawals and deposits. What if we extend the withdrawal functionality so we can also pay for blobs? Instead of scripting, we could have full programmability as a precondition for posting a blob while only verifying the zk-proof and not executing the precondition itself.

ZK-Proof of:

- I own a particular certificate or NFT that allows me to post from this Snacc
- I solved a challenging puzzle, POW leader selection
- I waited X amount of blocks without a stateroot update → Now I initiate a forced withdrawal
- This shared sequencer-set consensus round is valid. Snacc is a treasury of the shared sequencer

Many other ideas could be an excellent pre-condition for posting a blob that otherwise would not be possible. As rollup reading a namespace to distinguish the blob using a snacc, you could flip a bit as suggested in “namespace encoding” and add the “signer” to the first share of the blob. In this case, the signer is the snaccs verifying key. Now, the rollup can only read the shares posted with a snacc, making the fork-choice rule much more flexible and leveraging Celestia to verify the zkp.

The pattern should be the same as for normal interactions on Celestia’s state machine. Normally, an account posts a signature over the public key to prove ownership and eligibility to spend tokens. Now instead of a signature, you post a valid ZKP corresponding to the verification key. With that pattern, a snacc should be able to do what any normal account on Celestia can do, like posting a blob.

Intra namespace sorting & and sub-namespaces

As we saw in all three ideas, we can add a suffix to the namespace in the first share of that blob to distinguish the trait we are trying to show. However, this is still susceptible to the woods attack if the roll-up has to filter a lot of garbage before it can get to the blob it cares about.

We can circumvent this by sorting the namespace by the suffix that we added. Let’s call the suffix a sub-namespace. If we take the signer, for example, we would group all blobs by the signer of the blob. That way, when reading the namespace, a rollup could focus on the blobs at hand, read a range of blobs, and stop when finished. The woods attack has no chance of succeeding, as you cannot post to this range because you are not the signer.

As namespaces are sorted lexicographically in the square, let’s call the grouping and sorting of the sub-namespaces intra-namespace sorting.

In addition to changing it on the square level, the DA-Natwork could extend the API for certain intra-namespace sorting conditions and expose sub-namespaces for queries.

The extension of the API is possible in a trust-minimized way because we can create range-proofs over the blobs with specific signers if the namespace is sorted. In NMT range proofs, we check nodes on the left and right of the inclusion proof for completeness, such that we prove that we only have blobs from the namespace. We can apply the same pattern to sub-range proofs over two properties: the namespace and the signer. For that, we look at the left and right blob that is not in the range. From that, we only need the first share to determine that the subrange includes all blobs and that we are not missing any. (Note: we can be sure that the left share is the latest blob before the sub-range because we can calculate the distance to the blob using the length of the blob encoded in the share in addition to the padding using the NI-default rules applied to the first blob in the sub-namespace)

In addition to a sub-range proof, we can also have an absence proof. We need two consecutive blobs and proofs of their first shares for that. With that information, we can correctly check if they are consecutive, and we do that for signerA and signerB. Then, we can show that signerA < signerX < signerB with signerX being the signer we want to show an absence from for this specific namespace.

This idea works similarly for signers, script hashes, and snaccs account verification keys. We could also add other metadata to sort the namespace by, for example, fees paid for the blobs.

Sorting the namespace is like applying some forkchoice rule to it. If we extend his thought, we can leverage this sorting to benefit the rollup. Suppose we could define the priority of a rollup blob in a single number. If a snacc defines the fork choice rule, we could get an additional output in the verification being a priority number. That priority number could be another suffix to add and sort by. After that, the Rollup is potentially only required to read one blob with the highest priority. This thought is the least defined yet, so I welcome comments on whether this is something desired or if people could find applications for such a feature.

The final idea would be to include the transaction as an input to the ZKP used for the pre-condition. This would allow us to extract information like fees paid by the user, the account that is submitting the PFB transaction, the size of the blob, and more. It could go even further, looking into the blob’s contents and cross-checking it with the blob commitment. With that, you could add semantic checks to the transactions of the blobs or maybe even have more in-depth checks like gas accounting for the rollup.

I believe the end-game namespace would be sorted as indicated using namespace encoding, have a flexible size limit using a 1559-style controller, and have a snacc precondition before posting the blob.

Opening Pandoras Box

We touched on the idea that snaccs could post blobs to Celestia. If we adopt this idea, we should extend it to all Celestia transaction types. A snacc should be able to pay for gas for any transaction, such as staking or an IBC transfer. The precondition is a more abstract version of the fee grant & authz module, which is already in the cosmos-sdk and used by Celestia. A snacc would then also act as a super-set over ICA as you could do any header verification of the chain in a ZKP as it is done in Tendermint X reducing additional computation performed by Celestia.

If we go to the final step, a snacc should be able to deposit to another snacc. Having personal snaccs would be a way to create account abstraction that is out of the box natively on the base layer. Dedicated snaccs could also unlock huge UX improvements, like social recovery, privacy, and composability, by using Vitalik's idea of a keystore rollup on the base layer. A snacc could hold the user's keys and allow them to deposit to any other snacc. In a multi-rollup world, we want an easy way to do key rotation over ALL rollups we are using. We can achieve smooth key rotation using that keystore rollup where the key rotation happens on a snacc, but the resulting snacc interacting with other zk-rollups can stay the same.

Not only for the rollups' users but also for the users who regularly post blobs to Celestia, this could replace the feegrant module and give treasury management of rollups and service providers much more depth in customization and UX than feegrant and authz could not provide before.

The last major use case I see is for liquid staking and restaking. For liquid staking, the transition seems obvious in this case. Let's take Stirde, for example. Currently, Stride's user funds that stake to Celestia are held in a 5/7 multisig. That is the best that any liquid staking protocol can do right now. Now with CIP-14 and Strides proposal they plan to upgrade to an ICA account the moment ICA is live on Celestia. The next obvious step would be to move from an ICA to a snacc if it can interact with any transaction type on Celestia. This construction would also mean that rollups could stake users' funds after they deposited them to the rollup in a trust-minimized way. You can achieve that by baking in an undelegation to the withdrawal that you are doing from a rollup to the base layer.

Staking and slashing as a function can be done on a rollup, which means they can be done on a snacc. The advantage of doing it on a snacc is the trust-minimized bridge of TIA to the snacc to be able to use TIA for staking. If we combine the liquid staking snacc and the staking/slashing snacc you basically get an LRT where instead of staking TIA you would stake stTIA. Slashing here can be a combination of undelegation and burning or whatever ruleset the snacc decides on.

Snaccs depositing to each other might be the Pandora's box we want to open.

[

image

1024×1024 85.7 KB

](https://forum.celestia.org/uploads/default/original/2X/1/12b84a82053a67c12c0d83c4c8bc7d3ee7c634e7.jpeg)

Open Questions

- What bytes do we want to use for namespace encoding?
- What conditions do we want to encode?
- Do we need a script if we can use a snacc?
- How would we abstract the snacc design to perform these features?
- Is snacc a good name for snark-accounts? Disagreement must follow with an alternative.

I want to thank everyone who discussed this with me, including, but not limited to @evan @STB @adlerjohn @musalbas @Bidon15 @hxrts and the zk-working group. All bad ideas are mine only.