Sync-patterns for the OpenPrismNode

The OpenPrismNode is a custom and open-source implementation of the existing proprietary PRISM Node by IOG. Both applications read data from the blockchain, process it, and store it in a database to facilitate the lookup of DIDs. The additional capability to write DIDs to the blockchain is mostly unrelated to the read functionality and is not the topic of this research.

All this work is based on the published (https://github.com/input-output-hk/prism-did-method-spec/blob/main/w3c-spec/PRISM-method.md), which has evolved over the years. While PRISM v1 is still in use (on Mainnet and Preprod) and differs from PRISM v2 in terms of what data is written to the blockchain, the OpenPrismNode is only concerned with the PRISM v2 specification as v1 is no longer supported.

When a DID is written to the blockchain, this is done in an initial CreateDID operation. The operation is packed as a proto-buf object and then serialized as a JSON object, which is then written to the Cardano blockchain (only mainnet and preprod at this moment) as a metadata transaction. Metadata transactions have a label to make them more easily discoverable. In the case of the PRISM Metadata-transactions, this label is “21325”.

There are currently three types of DID operations: the previously mentioned CreateDID operation, the UpdateDID operation (with multiple sub-variants), and the DeactivateDID operation. Additionally, there is the UpdateProtocolVersion operation, which is less important and hasn’t been used so far, as I’m aware. With PRISM v1, there were additional variants of the previous operations around key revocation as well as operations to write and revoke credentials. These features will not be supported going forward. At first glance, this might indicate a feature reduction, but that’s not the case. Rather, it's a different implementation concept, with the realization that credentials shouldn’t be written to any blockchain at all—not even as a hash.

When parsing both the preprod and mainnet for these PRISM operations, one will encounter not only valid PRISM v1 and PRISM v2 operations but also invalid data structures and malformed DIDs, as well as invalid operations where signatures are incorrect or don’t match previously parsed information. It is therefore important to closely follow the specification and reject all non-valid operations.

While a new CreateDID operation can nearly always be written and parsed to the blockchain (as long as the DID doesn’t yet exist), UpdateDID operations always reference a previously made operation. Take the following example: A DID was initially written to the blockchain (C1). Shortly after that, this DID then gets updated (U1) and again updated a second time (U2), before it gets deactivated (D1). The operation U1 references the previously created C1. U2 references U1, and D1 references U2. Any update or deactivation that is out of order and unable to reference the previous operation of that same DID must also be rejected. This is get complex when multiple operations are inside of a single transation. The referencing for prior operations is mainly the reason why a database of the current state of all DIDs is needed. Each operation has to be compared to the database to determine if a new operation is valid or not. A simple lookup from, e.g., U2 to U1 without this dedicated database of the current state of DIDs is not possible since U2 is not directly referencing a specific block in which U1 resides but a hash of the operation, which has to be parsed and processed first.

This also means that the database of all DIDs has to be built up by scanning the complete chain from the beginning, inspecting every transaction (to check if it contains metadata with the correct label), and then adding only the valid ones to the database if they are compatible with the previous operations inside the database. This is inherently a synchronous process.

The existing implementation of the IOG PRISM node, as well as the new codebase of the OpenPrismNode, relies on dbSync to process all operations on-chain. Running a Prism Node therefore also means running a full cardano-node, as well as dbSync and the PostgreSQL database required for dbSync. All in all, this setup requires a non-trivial amount of computing resources, knowledge, and time to get a fully synced Prism Node (database) online. The question posed in this document is about how to improve on that situation, with the goal of making running a Prism Node more accessible and therefore more decentralized, without taking shortcuts on the required validity checks of a fully synced database of all PRISM-related operations.

1. What to Sync? Rollbacks and Transaction Data

As explained previously, all the PRISM-related operations are captured inside metadata transactions. After the Cardano node and dbSync are fully synced, it is possible to simply query the PostgreSQL database of dbSync to fetch only operations with related metadata. This SQL query can be constructed to retrieve those operations in the correct order they appeared on the blockchain and then process them in sequence. This is the most efficient way of getting operations from dbSync. The slower alternative would be going through each transaction of each block and inspecting the metadata. While the second approach is much slower, it also allows the capturing of all blocks, even if they don't relate to PRISM directly. This enables the construction of a simplified blockchain inside the node's database, where each block references the previous one.

Both approaches are generally valid and come down to the actual requirements and the feature set:

- In the first (direct) approach, the node database would only hold the PRISM-related operations and nothing else, other than a reference to the block and the transaction a certain operation is contained in.
- In the second approach, the node database would contain a linked list of all block IDs (not the contents of every block) as well as the PRISM operations.

The second approach would likely result in an overall database size at least one order of magnitude larger than the first approach, with the benefit of having a linked list, which makes handling and validation of rollbacks much easier – if not possible at all in the first place.

Rollbacks are events where the blockchain is rolled back for a few blocks and continues the chain with a previous block. This means that previously written blocks (and possibly PRISM operations) can become invalid and therefore have to be removed from the database. Shallow rollbacks of only a few blocks can happen multiple times a day and are part of normal operation, while deeper rollbacks occur relatively infrequently.

The official IOG PRISM node doesn't handle rollbacks directly and assumes a rollback-free environment by waiting a certain number of blocks until the chance of a rollback is negligible. This means that the processing of each PRISM operation takes a substantial amount of time. Depending on the level of certainity this can range from 1-2 minutes (as we have seen previously on preprod), up to multiple hours. The current configuration (see spec https://github.com/input-output-hk/prism-did-method-spec/blob/main/w3c-spec/PRISM-method.md) states that for mainnet the secure-depth should be 112 blocks, which equates to currently about 45-55 minutes (around 24s per block).

To improve on the potential wait time, I suggest implementing direct feedback to the user when writing an operation to the chain and returning the current block depth (as a gauge of the certainty of a successful operation) back to the user. This also means that the OpenPrismNode would not assume a rollback-free environment and therefore would have to keep track of the linked list of blocks to handle rollbacks gracefully. It is a trade-off between initial sync speed, disk space, operation certainty, and reaction time. Since the latter is closely related to usability, I lean heavily towards faster response times and handling the issue in the application itself rather than on the side of the node. For example, if a user creates a DID and then immediately issues credentials with this DID, a rollback of the CreateDID operation invalidates all issued credentials, as the issuing key of the DID issuing those credentials cannot be resolved. This is obviously a bad user experience but will happen only very rarely. To improve this situation, the application could check the current block height and compare it to the one of the last update of a DID before allowing it to issue a credential. This is preferable to letting the user wait for 45 minutes after every operation on the chain.

In this current model, only the relevant PRISM operations are captured and stored, but there is additional metadata available for each block or transaction that could also be stored to provide a fuller picture. This mainly includes the incoming and outgoing eUTXO transactions, which show who paid and therefore initiated the transaction. Depending on how the node was set up, this can indicate who operates a specific PRISM node. This is not necessary metadata, but it could provide interesting insights for statistical analysis. This is currently not done by the IOG nodes (as far as available information suggests) and might bring additional value. On the other hand, it makes the OpenPrismNode implementation more complex and increases the data footprint.

2. Data Providers

An alternative to running a full Cardano node and dbSync is using external data providers. These provide REST APIs to access the full dataset on-chain. Depending on the model chosen in section 1, these can be a viable alternative or costly and time consuming. The third-party data providers considered here are Blockfrost (blockfrost.io), Maestro (gomaestro.org), and Gimbalabs Dandelion (gimbalabs.com/dandelion).

The principle is generally the same: the data provider runs at least one Cardano node and dbSync and provides API access to the data inside the PostgreSQL database. To improve performance, a caching layer might be added by the data provider for faster access.

The APIs differ between vendors. In some cases, it is a one-to-one mirror of the dataset inside the PostgreSQL database, while in other cases, the queries can be more elaborate to capture complex data structures in a single request.

This approach comes with two main problems:

1. **High Number of Queries**: When the OpenPrismNode implementation tries to build a fully linked list (as described in section 1), a high number of queries are necessary: a single query

is usually required to get a block, a second query for transactions inside that block, and at least another one for the metadata. Depending on the API definition, this can result in a very high number of operations. Some optimizations can be implemented, such as first getting all the metadata with the PRISM-specific label, then going back to the containing block.

The preprod network currently contains around 2.3 million blocks. Mainnet has a block height of about 10.5 million blocks. Assuming a rough estimate of 1.5 requests per block to get all information, this amounts to 3.5 million requests to fully sync an OpenPrismNode for preprod and around 16 million requests to do the same for Mainnet. To keep the nodes running, around 100,000 requests would likely be necessary, with some headroom.

Looking at the pricing tables for Blockfrost, for example, this would cost 29 EUR a month to keep a node synced and initially 79 EUR to sync the node. Each pricing tier has a daily limit on how many requests can be sent. The 79 EUR plan allows for 1,000,000 requests per day, which would take about two weeks to sync Mainnet. The 29 EUR plan allows for 300,000 daily requests, which should allow someone to continue running two fully synced PRISM nodes (e.g., one for Mainnet and one for Preprod). While this is a reasonable offering as it frees one from the burden of running and upgrading a Cardano node as well as dbSync, the initial syncing time of around two weeks seems quite long. On top of that, occasionally shutdowns of a node (e.g. for updates) may require extensive resyncs, which may force the user to again use a higher tier.

The alternative would clearly be to switch to the simpler data model, in which the PRISM node would not contain a linked list of all blocks but only the PRISM transactions. The monthly costs of 29 EUR will still stay the same (because of other limitations of the free plan), but the initial syncing could likely be done in a few hours.

2. **Reliability and Trust Issues**: While self-hosting isn’t inherently more reliable (and might often be less so), there is often no clear insight into why a data provider (e.g., Blockfrost) is not responding or is offline or not providing the expected data. Another problem is trust. For example, resolving a DID to a DID-Document is a sensitive process when it comes to production data, and there might be an underlying reason to run a PRISM node in the first place—to have a fully trusted root. By introducing a dependency on an external provider at the lowest level, the trust chain might not be strong enough, depending on the reason for running a node in the first place. The current ecosystem is still to young to determine the actual need of PRISM node operators and the trust requirements.

3. Service Providers

An alternative to data providers could be Demeter (by TxPipe), which functions more as a service provider. Instead of providing a layer in between, it offers direct access to a Cardano node, a dbSync instance, as well as other potentially interesting services. The pricing is based on the connection and the time the instance is provided, rather than per request. While this generally seems to improve on the criticisms of data providers, the actual performance remains to be seen and cannot be determined just by looking at the service description. Therefore, it has to be evaluated at a later stage of the project (after Milestone 2), when syncing with an existing on-premise node can be benchmarked and then compared with Demeter's offering in terms of pricing, throughput, and uptime.

4. Other Alternatives

- **Kupo** (https://github.com/cardanosolutions/kupo) is a chain indexer for resolving outputs by addresses, policy IDs, or output references. This allows fetching required data from the Cardano node without running dbSync. Unfortunately, the current implementation does not index metadata directly, so it can’t be used in a way that provides any significant benefit. This may change in the future.
- **Oura** (by TxPipe) is an application that sits on top of Cardano nodes and analyzes the current tip. It uses webhooks to notify a registered application of a certain pattern it is looking for. Alternatively, custom sinks can be defined where the data can be sent. While this approach offers the possibility to quickly stream new blocks with matching metadata into a PRISM node application, it does not query the chain. Therefore, to use Oura, one would have to pre-sync the PRISM node database and then hook it up to the incoming stream. Since pre-syncing would require one of the previous techniques (either dbSync or a data provider), the benefit of introducing an additional application is unclear.
- **Scrolls** (by TxPipe) is a cache of on-chain operations. It crawls the complete chain for specific types of operations and then monitors the tip of the chain for these types. The currently supported operations include “utxo by address” or “balance by address”. Scrolls currently doesn’t support parsing metadata-related information and therefore can’t be used as the backbone for the OpenPrismNode. However, the project is under active development, and they plan to support different operations, such as “by metadata label”, which might be useful in the future. It might make sense to revisit this project at a later point to evaluate it again.

Summary

This document provides an early summary of the different challenges and approaches for syncing a PRISM node database. At the current stage, it seems sensible to follow the initial plan and first finish the integration with dbSync, as it offers all the benefits with the only drawbacks being technical complexity and required compute resources. In light of the required security and the need for a trusted solution, this seems reasonable, especially when the OpenPrismNode has to be hosted as a highly available service with a PostgreSQL database.

While the approach of using existing data providers like Blockfrost is a reasonable strategy for keeping a node up to date, it doesn’t address the issue of initial synchronization. Waiting multiple days and sending millions of HTTP requests is far from an optimal solution. The offering of Demeter, on the other hand, seems more trustworthy as it removes the middle layer and directly provides access to an underlying instance of dbSync without the need to run the complete infrastructure. Scrolls by TxPipe is also interesting, but the required features to use it as an alternative datafeed for the OpenPrismNode are not yet implemented and part of the road-map, so it has to be reevaluated at a later stage.