# Data Onramp with Filecoin Bridges

Nicola G Wyatt Daviau
2024 May

# Golas

1. **Exporting Filecoin services to other chains**: Data can be stored from other relevant L1s, L2s and other private services into Filecoin.
2. **Storing small data without effort**: Smaller than sector (and piece!) data can be stored into Filecoin with ease.

# Summary

The FVM brings smart contracts to filecoin allowing developers to do productive novel things with verifiable data. One obvious thing to do is allow clients to put data on filecoin more easily. Enabling onboarding small pieces is better for users but worse for scale which motivates bridging to higher throughput lower latency lower cost systems like blockchain L2s. Some redesigning of the standard client contract flow is needed for things to work on L2s. Essentially the contract should be split into two halves: an expensive contract onboarding, aggregating and paying for data and an efficient contract transmitting proof of data storage.

FilOz is proposing to build a prototype of a proposed architecture here: https://github.com/ZenGround0/onramp-contracts/tree/main so that developers can build applications more easily following this design.

We're currently prototyping. First getting it working with L1 and L2 == Filecoin, then getting it working with true bridging on calibration net. Once we're done we'll measure performance, gather feedback and check if it works as well as we think.

# Motivation

Here is a walk through the decision tree that got us where we are. You can skip this if you just want to know what we're building and not why.

## Why small pieces

We have an informed guess that web3 users are underserved because they are interested in smaller data (1 B - 1 GB) than is currently easy to onboard without centralized third parties.

## Why aggregation

Filecoin works on economies of scale and SPs have lower cost if they onboard larger chunks of data. So we want to provide larger deal sizes for SPs while providing smaller deal sizes for users. We have software to do this today in a trustless way with all of the prior work done with PODSI:
https://github.com/filecoin-project/FIPs/blob/master/FRCs/frc-0058.md

## Why Blockchain L2s

First blockchain L2s closely match our scaling needs trustless security model + programmable + higher throughput and already exist. Proposals to build offchain systems are feasible but require new types of operators. In particular (1) someone to run a server doing aggregation logic and (2) economic actors serving as payment network routers in the style of the lightning network. Both these are feasible but they are not the easiest or fastes way to get to trustless onboarding of small pieces. They require not just programming but business development to achieve basic operation.

Second blockchain L2s have organic activity that can get value from filecoin storage and so are good places to sell filecoin storage.

Note that the system described here will trivially work with any L1 and L2 that run evm contracts and can be connected by a bridging mechanism. This includes both L1 and L2 being Filecoin if scaling turns out to be less of a problem than anticipated.

## Why payments on L2s?

This design has all payments accounted for and settled on the L2. Again the reason for this is scaling. Half of the scaling issues are in managing piece cids the other half are managing payments. This is because with many small deals the system is now processing many small payments. We know how to build systems that allow for true native token settlement and they require building out a payment processing network like

the lightning network. Note that the design covered here works just as well with wrapped FIL on an L2 as any other token.

## Why use the deal system for data proofs?

Filecoin has deals and sectors. Sectors go into deals. Traditionally users only onboard data into deals using the deal system. The DDO FIP enables data onboarding directly into the sector system. However the DDO work is not finished and notifications to smart contracts during onboarding are not enabled. This design uses the deal system because it works today. The sector system would likely be cheaper as each deal per PublishStorageDeal method takes 100-200 M gas. But we won't block on designing and implementing relatively complex FIP when existing strategy works well enough.

## Why use client contract pattern and not deal bounty pattern?

When integrating the deal system into the FVM you can

1. make a deal directly with the contract (client contract)
2. incentivize other parties to make a deal and then provide deal as proof (deal bounty contract)

We are working with the client contract pattern because it reduces filecoin method calls. There are advantages to the deal bounty pattern but they mostly appear with a new set of economic actors (deal bounty hunters) in the network which don't yet exist.

## Existing work

The design came while studying @Lordforever's existing work on Eastore: https://github.com/Eastore-project/Eastore-Contracts/tree/master/contracts which organically included axelar bridging in a client contract. We hope our ideas will end up being used in products like Eastore.

# Design

# Protocol Design

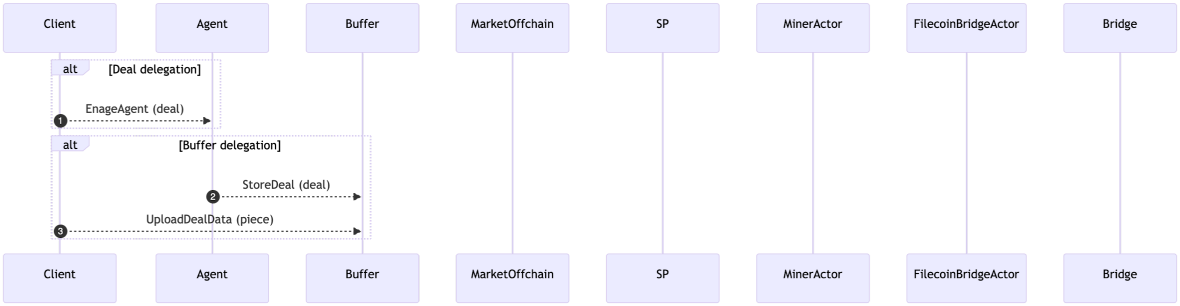
## Participants

- **Client**: A user that wants to store data.
- **Agent**: An agent that helps the user store the data on their behalf (optional).
- **Buffer**: A service that stores data before it gets stored into Filecoin (optional).
- **SP**: The storage provider that is offering storage services.
- **MarketOffchain**: The market where agents advertise deals and storage providers commit to them.
- **MinerActor**: The Filecoin actor that ensures that data is being stored and proven.
- **FilecoinBridgeActor**: An actor on L1 that sends messages to the bridge to speak to a L2/other chain.
- **Bridge**: A cross-chain bridge that bridges messages from Filecoin into other systems.

## Deal and Data Delegation protocols (optional)

This set of protocols are designed to make sure that the User can delegate their data and client making to third parties.

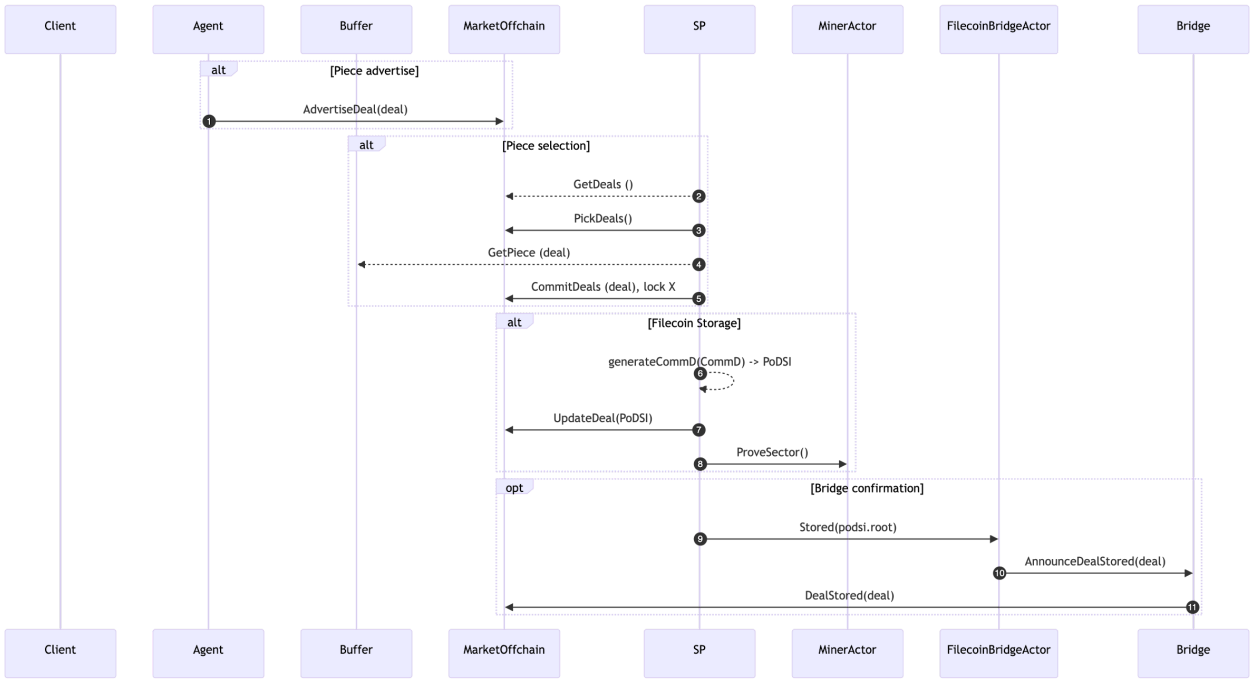
Note the client can also be the buffer and the agent, for simplicity and more modularity we divide their roles here.




## Filecoin storage bridge (mandatory)

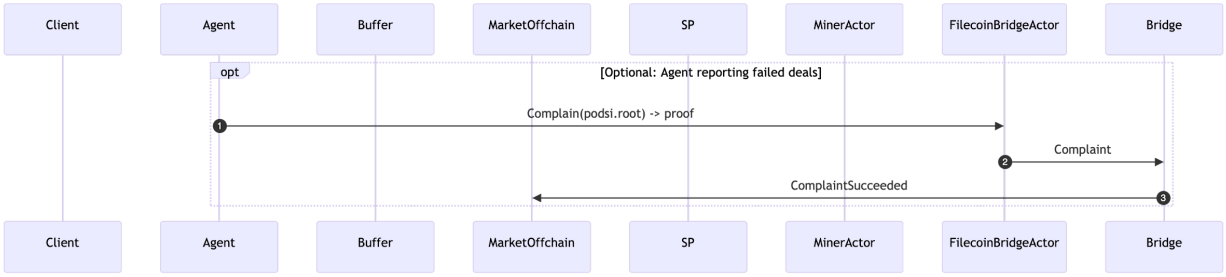
This set of protocols make the agent advertise the deal so that storage providers can commit to store it, store it into Filecoin and then using the bridge, show that the data has been stored.

This protocol only guarantees that the data has been stored, not that it is continuosly stored. (This is an equivalent to an HTTP POST)




## Report faults from Filecoin (optional)

This protocol makes sure that if the data is not stored, the agent can read Filecoin L1 and report this back into the offchain market.




## Implementation Design

Here is a high level overview of the flow of data and proofs:

Users submit information of data storage to an `OnRamp` contract. We'll call this information an `offer`. It is similar to a filecoin deal but does not need to not match filecoin deal structure beyond containing `commP`.

I am prototyping with this simple offer structure:

```
struct Offer {
```

```
    bytes commP;
    int64 duration;
    string location; // buffer uri
    uint256 amount;
    IERC20 token;
}
```

A client with a file to store prepares the data, gets commP, sets metadata and submits an `offer` to the `offer_data` method on the `OnRamp` contract. It transfers payment to the contract. It then emits a `DataReady` event. The client also moves its prepared data into a buffer where it can be fetched by any SP.

An SP running an L2 adapter on the L2 network hears the event, decides to store the file, retrieves from buffer. When it has enough files it does aggregation, generates PODSI proof and sends a message to `commitAggregate` method on OnRamp contract. `OnRampContract` checks PODSI proofs to ensure the aggregate piece cid contains all referenced offers. `OnRampContract` will now pay to the payout address send to `commitAggregate` when it gets proof of storage in filecoin network.

The SP on the filecoin network submits a filecoin deal for the aggregate piece to the `Prover` client contract. This filecoin deal has 0 payment or FIL+ attached as the incentive is on the L2. This triggers `Prover` `notify_deals` which is written to call to the `Oracle` contract on the L2 with the CommP being stored and relevant metadata such as duration. This is done with axelar.

The `Oracle` calls the `OnRamp` at `verifyDataStored` at which point the `OnRamp` will go through its accounting of all offers stored by the aggregate at the given CommP and process payments.

## Incomplete features and Known problems

### SP contention

Probably the biggest issue is that with many SPs competing for offers there is contention. All SPs will want valuable offers. A simple way to handle this with high costs is to allow SPs to commit to individual offers before aggregation. This way all contention happens before any transfer or aggregation work occurs. The issue with this is there will be many failed duplicated attempts to commit to an offer onchain (i.e. what happens with filecoin window post dispute).

One way to handle this is for clients to specify SPs. However this greatly complicates the user interface and limits the potential for contract behavior. This complexity should be handled by the onramp in all but special cases.

Another way to handle this is for contracts to run elections on which SPs get the opportunity to commit to an offer. This could be done round robin, randomly, or with arbitrary contract logic: example 1) SPs with high reputation scores get first dibs example 2) SPs do proper on chain auctions to claim offers.

In my opinion round robin elections for periods of time are the approach that is probably the easiest place to start and we're considering prototying this after we get the basics working.

In all cases, including the aggregate commitment described in the overview above, SPs that do not follow through on their commitment need to timeout and allow other SPs a chance to claim offers. This also isn't planned for the initial prototype but should be in any immediate followups.

### PODSI retrieval is broken

Empirically as discovered by @Lordforever:
https://filecoinproject.slack.com/archives/C03CKDLEWG1/p1707226315098939

We need to fix this for users to get data back at all with aggregation

### Faults not handled

The basic design enables paying for the creation of a filecoin deal. It does not expose errors (deal not published, deal terminated early) to the onramp contract. This will likely be important. To expose errors to the L2 we'll need another method on the prover contract that retrieves market actor deal status and reports to the oracle.

To get more accurate information about the status of data storage, i.e. temporary data faults, we'll need to move to using the sector system.

Perhaps a better way to handle this in the future is to delegate data storage to a repair system sitting on top of deals or sectors. This way all but the most catastrophic errors are shielded from the onramp contract and applications.

### Payments are one time

There are many ways to handle continued payments as in the filecoin deal system. One strategy is to push this complexity to the ERC20 used as payment. Continued payments will likely need to interact with fault reporting mechanisms.

### Buffer operation

Someone needs to run the buffer either client (IPFS) or third party. If its a reliable third party then they need to be paid. This ends up looking like requiring a version of the storage problem filecoin solves if we want this to be trustless. We could resort to federated or centralized solutions.

## Building Blocks

What we are building. Separated into evm contracts and "backends" -- regular software someone is running on a local machine

### OnRampContract

Runs on L2. Expensive logic in the limit where piece size is small. Take in offers from clients, emit events for SPs, receive data proofs and process payments. A good place for application logic for various different programmable data systems.

https://github.com/ZenGround0/onramp-contracts/blob/main/src/OnRamp.sol

### ProverContract

Runs on L1. Listens for data proofs and exports them back to the OnRamp via the Oracle.

https://github.com/ZenGround0/onramp-contracts/blob/main/src/Prover.sol

### OracleContract

Runs on L2. Gets verified data from L1 and pushes to the OnRamp. Many possible implementations. Today we're using axelar. In the future trustless F3 bridging would plug in here.

https://github.com/ZenGround0/onramp-contracts/blob/main/src/Oracles.sol

### ClientBackend

Offers local data to the filecoin network.

1. stream-commp + go-car
2. Build a tiny buffer wrapper
3. Build a tiny eth client wrapper

and glue these together

### BufferBackend

Temporary storage between client and SP

Start with local filesystem impl Then move to HTTP Might try out IPFS (client hosts until SP pulls)

### SPAdapterBackend

Code SP uses to listen for and satisfy data offers on an L2

A local daemon

1. uses ethclient to listen to events.
2. basic bin packing logic
3. aggregation and PODSI: https://github.com/filecoin-project/go-data-segment
4. sending commit message with ethclient
5. create proposal / data location and send to boost

If things go well this can be added directly to Boost.

# Implementation plan

1. Get everything working on L1 == Filecoin L2 == Filecoin on local machine devnet (May 2024)
2. Get everything working on L1 == Filecoin L2 == (Optimism | Polygon |...) using calibration net and axelar (May - June 2024)

# Onramp landscape

- Basin
- Web3.Storage
- Lighthouse
- Eastore

# Design decisions and learning

### Where do operations happen?

- Payments (most likely L2)
- Market (L2)
- Slashing (L2)

### Who verifies the deal of the correctness of the data structure?

- Publicly verifiable deal: the layer 2
- Privately verifiable deal: the client

We decided that deals are publicly verifiable deals

### Bridges vs Oracle

If we could replace the message passing (Axelar) with an oracle (Axiom), then any small chain/l2 not supported by Axelar will work.

Ideally we design for oracles and not for bridges.