Background

Phase 1 is currently being specified to fall in line with the simplified shard chain proposal released during Devcon 5. Phase 1's main value is bringing data availability to eth1 (without execution). This allows eth1 to make proofs against state roots from eth2's shard chains at a capacity of ~2-3 MB/s. In the world of optimistic rollups and zk rollups, this increased data capacity provided to eth1 gives us an early path to scalability.

In order to use phase 1 as a data availability layer, we need to provide the ability to upload data to the shards via a fee market. We also need to expose finalized state roots from eth2 to eth1 via a bridge. This writeup looks into various approaches and discusses their tradeoffs and requirements.

Eth1 - Eth2 Bridge and Integration

As stated in the background, we need some level of integration between eth1 and eth2 for phase 1 to be useful. Eth1 needs a way to reference or read the data from eth2's shards. We also need to discuss the relation between validator transfers, Beth <-> eth conversion and more.

Validator Transfers

In order to prevent fragmentation of eth and Beth (and other technical reasons we won't dive into for now), a two-way bridge supporting eth <-> Beth conversion is required before launching validator Beth transfers. This requirement affects the general roadmap.

Read-Only Bridge to Eth2 Without Conversion

A read-only bridge allows eth1 to access data from eth2's shards. We could implement the read bridge via the application or protocol layer. Casey describes how these may be implemented in this comment. To summarize, eth1 would be a light client of eth2 via one of these two methods:

Protocol Level

- eth1 miners would be required to run an eth2 light client
- ETH2_LAST_FINALIZED_BEACONBLOCKHASH

Opcode addition

- eth2BeaconBlockHash

added to the eth1 block header or a miner voting mechanism

- Hard fork

Application Level

- btcrelay style contract to track eth2 validator sets
- BLS12-381 precompile
- No hard fork

Finality Gadget

Vitalik describes how eth2 could expose a finality gadget for eth1. This would ensure the eth1 chain never re-orgs and takes a step forward on eth1 & eth2 integration. It also enables a faster and more secure bridging mechanism.

Beth <-> Eth Conversion

We may extend the models above by adding the ability to convert eth and Beth. This adds further complexities by requiring changes to the eth1 account trie and EVM. It also would likely necessitate the finality gadget.

Eth1 <-> Eth2 Merge Strategy

Vitalik described this strategy in detail recently. It brings eth2's POS mechanism to eth1 and makes eth1 a specialized shard in the eth2 ecosystem. This approach requires eth1 to first make its switch to a stateless protocol (although miners and Block Producers would still hold eth1 state in the interim). It also seems to make eth2 more eth1 centric (or as an extension of eth1). One previously undiscussed benefit is that eth1 could store its witnesses/proofs in the other shards.

Fee Market & Data Upload Methods

Below are methods on how to accomplish user data uploads into eth2's shard chains.

Validator Uploads

Vitalik described this approach in a comment. In order to pay fees, you would need to have eth in an eth2 validator account. These models assume an operator aggregates user transactions (off-chain), so would only need one fee payment/upload per rollup block. The process goes as follows:

1. Operator deposits eth on eth1 to deposit in an eth2 validator account.
2. Eth arrives in a validator account on eth2.
3. Validator provides a fee signature (which is a BLS signature of [data_root, shard, slot, fee]

) via a simple mempool or communication channel with the Block Producer (BP) along with the requested data blob aggregated from user transactions.

1. The shard BP verifies the validator account balance on the beacon chain, includes the data in the block body, and adds the fee signature to the block header.
2. The beacon chain has a new operation called process_data_fees

which pays the BP the fee from the validator's balance. This operation would require a DataFee

container with a proof of data inclusion from the shard block.

Pros

- Simple workflow and can build on patterns from other beacon operations
- Reduced complexity
- Scope could be reduced due to only needing a read bridge without Beth <-> Eth conversion

Cons

- Increased load on the beacon chain if there are frequent uploads (although seems unlikely)
- Does not map an upload account to an eth1 account (needs a validator account)
- Does not necessarily move progress forward on efforts towards phase 2 or an eth1 <-> eth2 merge strategy
- process_data_fees

could be considered a roundabout way to support Beth transfers. We'd like to not support Beth transfers until a two-way eth <-> Beth conversion is established. Limiting the fee amount to block_body_price

as described in the phase 1 spec could mitigate this issue.

Data Bounty

Casey originally discussed this approach in a comment. It is similar to the process above and also assumes an operator aggregates user transactions (off-chain). As a result, there would only be one fee payment/upload per rollup block. The process is as follows:

1. Operator deposits eth into a data bounty contract on eth1 offering a bounty [data_root, fee]
2. Operator sends the data and bounty details to the eth2 block producer via a direct communication channel or mempool
3. If the block producer on eth2 includes the data, the block producer may collect payment directly from the data bounty contract on eth1 by submitting a proof. The read bridge (discussed in bridge section) provides the ETH2_LAST_FINALIZED_BEACONBLOCKHASH

that the proof is verified against in the contract

Pros

- No need for beacon chain operations
- It does not surface a roundabout way to introduce Beth transfers
- Operators pay directly from their eth1 account
- Reduced complexity
- Scope could be reduced due to only needing a read bridge without Beth <-> Eth conversion

Cons

- Fragments block producer rewards between eth1 and eth2
- Extra contract logic on eth1
- Does not necessarily move progress forward on efforts towards phase 2 or an eth1 <-> eth2 merge strategy

Minimal Phase 2 (Phase 1.5)

This describes a reduced scope phase 2 which launches with the minimum necessary requirements to support a fee market.

Requirements

1. Simple transfers and fee market EE (multiple have already shown necessary performance)
2. Simplified relay/state provider market (no need for contract code, just transfer witnesses)
3. Restrict EE deployments

Data Upload Workflow

Acquire eth in the transfers EE and submit an upload transaction via a simple relay or state network.

Pros

- Moves us towards a more complete phase 2 via a minimalistic launch (iterative vs. waterfall approach)
- Enables transfers between accounts on eth2 and removes the need to support validator transfers on the beacon chain
- Focuses our efforts if we pursue an eth1 <-> eth2 merge strategy without first supporting a finality gadget in eth1

Cons

- More complexity
- Requires 2-way bridge or Beth <-> eth conversion which is a pre-requisite for transfers and therefore may have increased scope
- Fragments operator balance between eth1 and eth2
- Relay/state provider complexity (although simplified significantly)

More Complete Phase 2

Pros and Cons are similar to the Phase 1.5 approach, however eth2 is closer to a complete state. In this approach, we assume we have strong tooling on deploying and writing EEs (including contract EEs). We'd also have a fee market established on EE deployment with a more sophisticated relay/state provider market. The fee market EE may undergo upgrades/evolutions and we should see innovation around contract EEs.

Conclusion and Recommendation

If pursuing a read bridge without conversion will reduce scope in the nearterm, launching a fee market via the validator upload or data bounty methods could make the most sense. However, it should be properly scoped as building the read bridge could be more complex than anticipated. Also, it does not move development closer to the full eth1 <-> eth2 integration. Some of the required effort could be throwaway.

Moving towards the full eth1 <-> eth2 integration is the end goal (and would be the next step if pursuing the nearterm effort

described above). There could be an argument for disregarding the near term strategy in an effort to maintain focus and avoid throwaway work or scope creep. If the near term strategy is disregarded, a limited phase 2 may be the best option for the phase 1 fee market. Later, a transition could be made to a more complete phase 2. Another question - is a finality gadget still needed for eth1 or is this now obsolete in the new integration?