PeerDAS

This is an sketch representing the general direction of PeerDAS

. This is being circulated at an early stage for feedback before further discussion refinement.

This set of ideas came out of conversations with Dankrad, Vitalik, members of Codex, RIG, ARG, and Consensus R&D. Directionally, pieces of this type of approach have also been under discussion in various avenues for the past couple of years, e.g. Proto's PeerDHT

The intent of a PeerDAS design is to reuse well known, battle-tested p2p components already in production in Ethereum to bring additional DA scale beyond that of 4844 while keeping the minimum amount of work of honest nodes in the same realm as 4844 (downloading < 1MB per slot).

This is an exploration to better understand what scale we can get out of a relatively simple network structure with various distributions of node types without

relying on a more advanced DHT-like solution.

Simulations of effectiveness of such a solution involve parametrizing the following:

- data size (number of rows and columns per block, as well as sample size)
- total number of number of nodes on the network
- minimum amount of work an honest node is expected to do (e.g. custody and serve samples from X rows and columns)
- the distribution of node capacities (i.e. what fraction of the network is minimally honest (custodies/serves X) and beyond (e.g. custodies/serves 10%, 25%, 100% of network data)) and the max sample requests they are willing to support (i.e. Y samples per slot)
- honest/byzantine ratio assumptions

Different parametrizations of the above will lead to either acceptable or broken configurations (e.g. data size might be too large to find peers of enough capacity for a given network distributions).

A note on DAS in general

Importantly, any DAS solution will rely upon nodes of various types and the assumptions we can make about them. Node types worth considering in any solution are:

- Validator and user nodes (Honesty custody/serve assumption can be placed upon them, e.g. download and serve samples of X rows/columns. Note, validators can be incentivized to custody but not necessarily to serve)
- High capacity nodes (some % of the data beyond baseline honest node)
- Super-full nodes (100% of data) [special case of high capacity node]

The difference between DAS solutions then becomes – how does the DAS network organize itself, how do you discover peers for sampling, how does the network utilize (or under-utilize) nodes of higher capacity (e.g. can the solution support lumpiness in node capacity or do all nodes look equal).

Sketch of PeerDAS

Configuration

The following is a bit of a reduction of the full requisite parametrization for illustration purposes.

All values in the table are example values and do not represent suggestions for an actual parametrization.

Name

Sample Value

Description

NUMBER_OF_ROWS_AND_COLUMNS

32

SAMPLES_PER_ROW_COLUMN

512

CUSTODY_REQUIREMENT

2

Minimum number of both rows and columns an honest node custodies and serves samples from

SAMPLES_PER_SLOT

75

Number of random samples a node queries per slot

NUMBER_OF_PEERS

70

Minimum number of peers a node maintains

How it works

Custody

Each node downloads and custodies a minimum of CUSTODY_REQUIREMENT

rows and CUSTODY_REQUIREMENT

columns per slot. The particular rows and columns that the node is required to custody are selected pseudo-randomly (more on this below).

A node may

choose to custody and serve more than the minimum honesty requirement. Such a node explicitly advertises a number greater than CUSTODY_REQUIREMENT

via the peer discovery mechanism – for example, in their enr (e.g. cust: 8

if the node custodies 8 rows and 8 columns each slot) – up to a maximum of NUMBER_OF_ROWS_AND_COLUMNS

(i.e. a super-full node).

A node stores the custodied rows/columns for the duration of the pruning period and responds to peer requests for samples on those rows/columns.

Public, deterministic selection

The particular rows and columns that a node custodies are selected pseudo-randomly as a function of the node-id, epoch, and custody size (sample function interface: custodied_rows(node_id, epoch, custody_size=CUSTODY_REQUIREMENT) -> List(uint64)

and column variant) – importantly this function can be run by any party as the inputs are all public.

Note

: increasing the custody_size

parameter for a given node_id

and epoch

extends the returned list (rather than being an entirely new shuffle) such that if custody_size

is unknown, the default CUSTODY_REQUIREMENT

will be correct for a subset of the node’s custody.

Note

: Even though this function accepts epoch as an input, the function can be tuned to remain stable for many epochs depending on network/subnet stability requirements. There is a trade-off between rigidity of the network and the depth to which a subnet can be utilized for recovery. To ensure subnets can be utilized for recovery, staggered rotation needs to happen likely on the order of the prune period.

Peer discovery

At each slot, a node needs to be able to readily sample from any set of rows and columns. To this end, a node should find and maintain a set of diverse and reliable peers that can regularly satisfy their sampling demands.

A node runs a background peer discovery process, maintaining at least NUMBER_OF_PEERS

of various custody distributions (both custody_size and row/column assignments). The combination of advertised cust

size and public node-id make this readily, publicly accessible.

NUMBER_OF_PEERS

should be tuned upward in the event of failed sampling.

Note

: while high-capacity and super-full nodes are high value with respect to satisfying sampling requirements, a node should maintain a distribution across node capacities as to not centralize the p2p graph too much (in the extreme becomes hub/spoke) and to distribute sampling load better across all nodes.

Note

: A DHT-based peer discovery mechanism is expected to be utilized in the above. The beacon-chain network currently utilizes discv5 in a similar method as described for finding peers of particular distributions of attestation subnets. Additional peer discovery methods are valuable to integrate (e.g. latent peer discovery via libp2p gossipsub) to add a defense in breadth against one of the discovery methods being attacked.

Row/Column gossip

There are both NUMBER_OF_ROWS_AND_COLUMNS

row and NUMBER_OF_ROWS_AND_COLUMNS

column gossip topics, one for each row/column – column_X

and row_Y

for X

and Y

from 0

to NUMBER_OF_ROWS_AND_COLUMNS

(non-inclusive).

To custody a particular row or column, a node joins the respective gossip subnet. Verifiable samples from their respective row/column are gossiped on the assigned subnet.

Reconstruction and cross-seeding

In the event a node does not receive all samples for a given row/column but does receive enough to reconstruct (e.g. 50%+, a function of coding rate), the node should reconstruct locally and send the reconstructed samples on the subnet.

Additionally, the node should send (cross-seed) any samples missing from a given row/column they are assigned to that they have obtained via an alternative method (ancillary gossip or reconstruction). E.g., if node reconstructs row_x

and is also participating in the column_y

subnet in which the (X, Y) sample was missing, send the reconstructed sample to column_y

.

Note

: A node is always maintaining a matrix view of the rows and columns they are following, able to cross-reference and cross-seed in either direction.

Note

: There are timing considerations to analyze – at what point does a node consider samples missing and chooses to reconstruct and cross-seed.

Note

: There may be anti-DoS and quality-of-service considerations around how to send samples and consider samples – is each individual sample a message or are they sent in aggregate forms.

Peer sampling

At each slot, a node makes (locally randomly determined) SAMPLES_PER_SLOT

queries for samples from their peers. A node utilizes custodied_rows()

and custodied_columns()

to determine which peer(s) to request from. If a node has enough good/honest peers across all rows and columns, this has a high chance of success.

Upon sampling, the node sends an DO_YOU_HAVE

packet for all samples to all peers who are determined to custody this sample according to their custodied_rows

/custodied_columns

. All peers answer first with a bitfield of the samples that they have.

Upon receiving a sample, a node will pass on the sample to any node which did not previously have this sample, known by DO_YOU_HAVE

response (but was supposed to have it according to its custodied_rows

/custodied_columns

).

Peer scoring

Due to the deterministic custody functions, a node knows exactly what a peer should be able to respond to. In the event that a peer does not respond to samples of their custodied rows/columns, a node may downscore or disconnect from a peer.

Note

: a peer might not respond to requests either because they are dishonest (don't actually custody the data), because of bandwidth saturation (local throttling), or because they were, themselves, not able to get all the samples. In the first two cases, the peer is not of consistent DAS value and a node can/should seek to optimize for better peers. In the latter, the node can make local determinations based on repeated DO_YOU_HAVE

queries to that peer and other peers to assess the value/honesty of the peer.

DAS providers

A DAS provider is a consistently-available-for-DAS-queries, super-full (or high capacity) node. To the p2p, these look just like other nodes but with high advertised capacity, and they should generally be able to be latently found via normal discovery.

They can also be found out-of-band and configured into a node to connect to directly and prioritize. E.g., some L2 DAO might support 10 super-full nodes as a public good, and nodes could choose to add some set of these to their local configuration to bolster their DAS quality of service.

Such direct peering utilizes a feature supported out of the box today on all nodes and can complement (and reduce attackability) of alternative peer discovery mechanisms.

A note on fork choice

The fork choice rule (essentially a DA filter) is orthogonal to a given DAS design

, other than the efficiency of particular design impacting it.

In any DAS design, there are probably a few degrees of freedom around timing, acceptability of short-term re-orgs, etc.

For example, the fork choice rule might require validators to do successful DAS on slot N to be able to include block of slot N in it's fork choice. That's the tightest DA filter. But trailing filters are also probably acceptable, knowing that there might be some failures/short re-orgs but that it doesn't hurt the aggregate security. E.g. The rule could be – DAS must be completed for slot N-1 for a child block in N to be included in the fork choice.

Such trailing techniques and their analyiss will be valuable for any DAS construction. The question is — can you relax how quickly you need to do DA and in the worst case not confirm unavailable data via attestations/finality, and what impact does it have on short-term re-orgs and fast confirmation rules.

TODO: Simulation and Analysis of PeerDAS

The crux of PeerDAS is that each node can find/maintain enough peers of enough capacity at each slot to meet their sampling requirements.

Thus for a given data size, we can easily simulate network distributions (node count, distributions of node type/capacity, number sample-requests each node can respond to, minimum number of peers, etc) to understand the chances of successful DAS at each slot.

We can calculate/simulate safe data sizes of various parametrizations and assumptions, e.g.:

- a homogeneous network of minimally honest nodes at a roughly 4844 custody requirement
- the above but then accented with 100 super-full nodes with X capacity for queries
- all sorts of distributions in between

Next up – simulations of various network distributions to anchor our understanding of the bounds of this method.