Network Shards

Note

: this is the high level idea on how the networking might potentially be organized to meet the future data sharding needs. This idea is kind of follow up to PeerDAS and SubnetDAS proposals. In some aspects it complements and in other aspects it serves as an alternative to these proposals.

The very basic idea is to split the network onto N

(let's assume N = 32

as initial approach) Network Shards

and let every shard take care of

- disseminating

(Push

) of 1/N

of the whole Gossip network traffic by serving as a backbone for various subnets (e.g. attestation, sync committee and future DA sampling subnets)

- custodying and serving

(Pull

) of 1/N

of the network data (DA samples, blob slices, blocks potentially)

Data dissemination (Gossip subnet backbones)

This idea might be thought of as a generalizedAttnet Revamp spec PR

Similar to Attnet Revamp every node in the network at any moment is assigned to a single Network Shard. A node serves as a Gossip backbone for a set of Gossip subnets statically assigned to this shard. Also similar to Attnet Revamp nodes are circulating across Network Shards in a deterministic manner.

The major advantage of this concept is the ability to uniformly and securely support Gossip subnets of a smaller sizes (even the corner case with a single publisher and a single subscriber). The concept of Network Shards also settles up another abstraction layer for push/pull data dissemination

Note

: A single Gossip subnet (topic) may potentially span several shards (the obvious example case is the beacon_block

topic which spans all shards)

Data custody

Together with serving Gossip backbones assigned to a shard the node also commits to custody and serve the data published on the shard topics. The data retention policy should be topic specific.

When a peer joins or reconnects to the network it should fill the custody gaps of missing past data to honestly fulfill its duties.

Note

: as the nodes are circulating across shards over time it's not that straightforward to retrieve historical data as a client. Different implementation strategies could be utilized to optimize this process.

Voluntary Shard participation

A node with high bandwidth and storage capabilities may voluntary want to join to more than one Network Shard. That could potentially be implemented by the method proposed in the PeerDAS write up

Danksharding application

Issues of existing approaches

- Finding and connecting to an abstract subnet is basically slow and takes unpredictable amount of time
- Small subnets are vulnerable to sybil attacks
- Pulling random samples (while catching up head) is also slow and unpredictable

Push sampling

Let’s consider the original Danksharding DAS (Data Availability Sampling). Every slot 256K

(512 * 512

) of data samples need to be published. Every node needs to receive just 75 of them (selected randomly by the node). Ideally a node should verify a different random subset of samples at every slot (or at least every few slots).

It is possible to have 256K

sample subnets split across all shards (e.g. 8K

subnets per shard).

A sampling node (from a ‘client’ perspective) should just maintain stable and balanced connections to nodes from all shards.

Note

: the above requirement to be connected to all shards could be relaxed for a regular node if the sampling algorithm could be relaxed: for example a node may randomly choose and slowly rotate the subset of shards, and randomly choose a sample subset from those sample subnets assigned to the chosen shards. However security properties need to be revisited for any relaxed approach.

A node would be able to Subscribe/Unsubscribe

corresponding sample subnets almost immediately

since there is no need to search and connect to subnet nodes

This concept meets the needs of various sampling approaches including original Danksharding,SubnetDAS approach

Pull sampling

Pulling recent samples is pretty straightforward: using a specific RPC method samples are requested from the nodes assigned to the corresponding shards.

Pulling historical samples is a bit more tricky due to shard nodes rotation. However various strategies may be used to optimize the process:

- Retrieve the samples which are available with the current connected peers while searching and connecting to the peers for missing samples
- Probably employ a more lenient sampling strategy with slightly relaxed security properties

Open questions

- (technical) Are Gossip implementations able to handle that number (order of 10K) of topic subscriptions?
- (Gossip protocol

change) topic wildcards? Aka das_shard_N_sample_*

- (Gossip protocol

change) topic hierarchy? Aka das/shard_N -> das/shard_N/sample_M

- (Gossip implementation

change) on demand subscription? Shard nodes subscribed to a single subnet das_shard_N

but if a client initiates a subscribe das_shard_N_sample_M

message then the node responds back the same subscribe

message

- (Gossip protocol

change) topic wildcards? Aka das_shard_N_sample_*

- (Gossip protocol

change) topic hierarchy? Aka das/shard_N -> das/shard_N/sample_M

- (Gossip implementation

change) on demand subscription? Shard nodes subscribed to a single subnet das_shard_N

but if a client initiates a subscribe das_shard_N_sample_M

message then the node responds back the same subscribe

message

- (technical) When a node (as a client) subscribes to a topic what is the probability to be included to the mesh promptly? Else a client would get message via gossip only (around 500ms of extra delay)
- Add staggering to nodes rotation across shards (has being discussed while coming up withAttnet Revamp)
- Add randomness to nodes rotation across shards such that it is only possible to predict shard assignments for the next M

epochs. This would help to mitigate sybil attacks on a single shard (a node which is live in discovery for more than M

epochs is guaranteed to be not specially crafted for a sybil attack) (has being discussed while coming up withAttnet Revamp)

- Number of shards:
- Cons of a smaller number of shards (more nodes per shard)
- Higher throughput and CPU load per node
- Larger custody storage per node
- Higher throughput and CPU load per node
- Larger custody storage per node
- Cons of a greater number of shards (less nodes per shard)
- Less reliable
- More vulnerable to attacks (sybil/eclipse)
- Higher number of peer connections for a client node which needs to be connected to all shards (e.g. for full sampling)
- Less reliable
- More vulnerable to attacks (sybil/eclipse)
- Higher number of peer connections for a client node which needs to be connected to all shards (e.g. for full sampling)
- Cons of a smaller number of shards (more nodes per shard)
- Higher throughput and CPU load per node
- Larger custody storage per node
- Higher throughput and CPU load per node
- Larger custody storage per node
- Cons of a greater number of shards (less nodes per shard)
- Less reliable

- More vulnerable to attacks (sybil/eclipse)
- Higher number of peer connections for a client node which needs to be connected to all shards (e.g. for full sampling)
- Less reliable
- More vulnerable to attacks (sybil/eclipse)
- Higher number of peer connections for a client node which needs to be connected to all shards (e.g. for full sampling)