Survey

https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/8954616#

s. Therein, one of the most practical methods to achieve horizontal scalability along with the increasing network size is sharding, by partitioning network into multiple shards so that the overhead of duplicating communication, storage, and computation in each full node can be avoided

it only provides a vague introduction of Ethereum 2.0, and the same problem exists in [43] where the consensus layer is decoupled from the ledger topology layer (which is inappropriate due to the importance of intra-consensus in a sharding system)

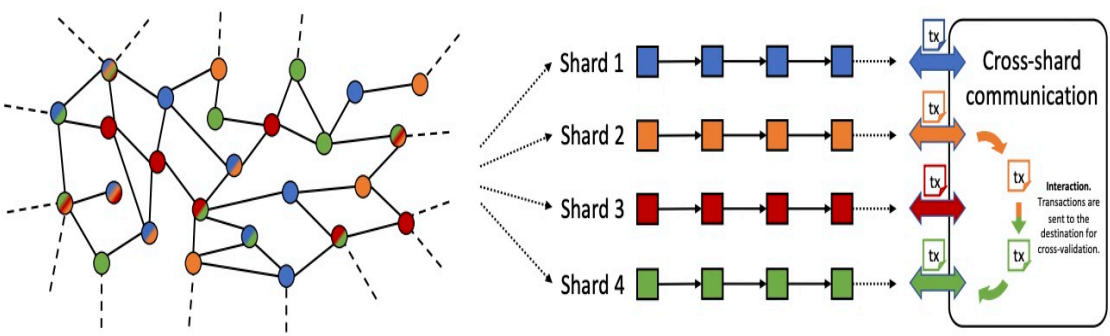

FIGURE 1. The sharding technology partitions the network into different groups, while each of the groups maintains its own ledger and processes and stores a disjoint set of transactions. By implementing a secure cross-shard communication protocol, such disjoint transaction sets that could not have been interacted become securely verifiable and interactively executable in parallel. Note that, nodes in some sharding mechanisms (e.g., Monoxide) can choose to participate in the processing of multiple shards and maintain their ledgers, as illustrated by the multicolored circles, while the unicolored circles denote the nodes only participating in a single shard to which they are assigned in terms of the color.

Summary of Survey

Ledgerless shard ?

based consensus algorithms [64], e.g., PBFT [5]).

Then it ends up encountering the dilemma of BFT-based 1% attack that the weak scalability of BFT-based consensus algorithm restricts the shard size, i.e., the number of members in a shard, while too small a size can potentially decrease the security of the intra-consensus with a strict fault-tolerance (FT), as described by the following cumulative binomial distribution,

$$s(k, m, p) = P[X \leq c] = \sum_{k=0}^{c} \binom{m}{k} p^k (1-p)^{m-k},$$
$$f(k, m, p) = 1 - s(k, m, p), \tag{1}$$

where X is the random variable that represents the number of times a malicious miner is picked [13], [57], [58], [65]; m denotes the shard size; c denotes the number of malicious members within a shard; and p denotes the total FT among the entire network. It is strongly suggested that $s(k, m, p)$ should be greater than

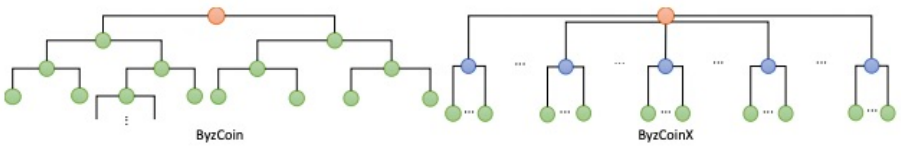

FIGURE 2. (Left) ByzCoin implements a tree with a fixed branching factor and an increasing depth. (Right) ByzCoinX implements a shadow tree with a fixed depth and an increasing branching factor.

https://hackmd.io/@HWeNw8hNRimMm2m2GH56Cw/sharding_proposal

A Explication of a sharding with samples by ethereum

Early ideas of samples

https://arxiv.org/pdf/1809.09044.pdf

Sharding with the fork choice rule

Beacon chain fork choice rule

The beacon chain fork choice rule is a hybrid that combines justification and finality with Latest Message Driven (LMD) Greediest Heaviest Observed SubTree (GHOST). At any point in time, a validator `v` subjectively calculates the beacon chain head as follows.

- Abstractly define `Store` as the type of storage object for the chain data, and let `store` be the set of attestations and blocks that the validator `v` has observed and verified (in particular, block ancestors must be recursively verified). Attestations not yet included in any chain are still included in `store`.
- Let `finalized_head` be the finalized block with the highest epoch. (A block `B` is finalized if there is a descendant of `B` in `store`, the processing of which sets `B` as finalized.)
- Let `justified_head` be the descendant of `finalized_head` with the highest epoch that has been justified for at least 1 epoch. (A block `B` is justified if there is a descendant of `B` in `store` the processing of which sets `B` as justified.) If no such descendant exists, set `justified_head` to `finalized_head`.
- Let `get_ancestor(store: Store, block: BeaconBlock, slot: Slot) -> BeaconBlock` be the ancestor of `block` with slot number `slot`. The `get_ancestor` function can be defined recursively as:

```python
def get_ancestor(store: Store, block: BeaconBlock, slot: Slot) -> BeaconBlock:
    """
    Get the ancestor of ``block`` with slot number ``slot``; return ``None`` if not found.
    """
    if block.slot == slot:
        return block
    elif block.slot < slot:
        return None
    else:
        return get_ancestor(store, store.get_parent(block), slot)
```

Helper struct that is used to encode/decode the state of the `ForkChoice` as SSZ bytes.

in other terms we see that Reed Salomon and other stark thing could help to do the encode/decode stuff!

Interop Metrics

The following are the minimal metrics agreed to be conformed upon prior to initial client interop efforts.

Name	Metric type	Usage	Sample collection event
`libp2p_peers`	Gauge	Tracks number of libp2p peers	On peer add/drop
`beacon_slot`	Gauge	Latest slot of the beacon chain state	On slot
`beacon_head_slot`	Gauge	Slot of the head block of the beacon chain	On fork choice
`beacon_head_root`	Gauge	Root of the head block of the beacon chain*	On fork choice
`beacon_finalized_epoch`	Gauge	Current finalized epoch	On epoch transition
`beacon_finalized_root`	Gauge	Current finalized root*	On epoch transition
`beacon_current_justified_epoch`	Gauge	Current justified epoch	On epoch transition
`beacon_current_justified_root`	Gauge	Current justified root*	On epoch transition
`beacon_previous_justified_epoch`	Gauge	Current previously justified epoch	On epoch transition
`beacon_previous_justified_root`	Gauge	Current previously justified root*	On epoch transition

In some way a only start to imagine a transversal sample overwhelming in Eth 2.0 ?

Hint:

```
/// Produce an `Attestation` that is valid for the given `slot` and `index`.
    ///
    /// Always attests to the canonical chain.
```