Here's an overview of Near's latest sharding design. We call it NightShade.

It's very different from what Near was doing before, it's more recent and thus not as well thought of yet, so some glaring holes might be present.

The goal of the design is to have a coherent approach with good data availability and atomicity guarantees, and a low chance of invalid state transition even with the adaptive adversary.

Block production

The first role that exists in NightShade is block producers. There is a limited number of them, and for the simplicity of modeling we say that there are as many block producers as there are shards. Say there's 100 of both.

Each block producer is assigned to several shards, say 4. Thus, each shard has 4 block producers assigned. Block producers assigned to a particular shard are known, and can somewhat easily be corrupted, by design. The block producers are assigned to the shard for one epoch, which let's say is approximately one day long. How exactly block producers rotate between epochs is orthogonal to the design.

Instead of modeling the entire system as one beacon chain and several shard chains, we model it as a single blockchain, that we call the main chain, and the blocks in the main chain contain the transactions and the state of the entire system. However, nobody in the system ever observes the full block, instead only the header of the block is distributed, that contains the merkle root of the transactions in the block, and the merkle root of the state before the transactions were applied, as well as some consensus information.

We call the parts of the block that contain the actual transactions applied in each shard in a particular block chunks. Each block has one (or zero) chunk for each shard.

The block producers need to collectively reach the following goals:

1. Collectively produce blocks on the main chain;
2. Prevent forks;
3. Ensure that the state transition function is applied correctly;
4. Ensure that all the chunks are available for some time after the block is produced.

Note that since the block producers can be corrupted adaptively, they alone cannot ensure any of the security properties (2-4 above), and can stall the block production. We will address the security below, and will try to make the stalling economically unreasonable.

The way block production works is relatively simple from here. For each height h of the main chain and each shard exactly one block producer is responsible for producing the chunk for that shard at that height. The four block producers assigned to a particular shard for the epoch take turns producing chunks. Once a block for height h is produced, each block producer downloads the chunks for the all the shards they are assigned to in the epoch, update the local state, and then create a chunk for the shard they are responsible for at the height h+1.

[

1344×802

]
(https://lh5.googleusercontent.com/SfnbXnEwJ0J6xXEVqcAWOpKDZMhXl8LpHMnQzcZ6aAcXFuk0adu5P8Fa5guVdew4JnI-zs364hsmmv9-E7464zhzfiK69mKlG1JYlzzsZCXXhSw7TOR7j__gehz1k14zC2a6dV6Y)

On the figure above each cell shows the shard for which the particular block producer produces a chunk at each height, covering the first 8 block producers with 4 block producers per shard.

Data Availability

Once a particular block producer created a chunk, they create an erasure-coded version of it with (n, f+1) code where n=3f+1 is the total number of block producers. The block producer then distributes the erasure-coded parts one to each other block producer. This way, for as long as f+1 block producers are online and cooperate, any chunk can be reconstructed. This idea is also inspired by Polkadot design, though we omit the data availability game they have on top of this.

[

998×646

](https://lh4.googleusercontent.com/4PZ1Dr7nHn-UXW7Ila5fKUFM_qEbviWtAZX-_IbNJCkDKRMXytwG4ILUqMYM-v7un5A3O32Bkq6jITu_onPlps_H7U1leh3Zd9VbWEz1RW3sHScYJUf826Dy_l3ED0gGJXkdY7o7)

Once the chunks are created and distributed, the block producers need to collectively agree on what chunks are available and can be included into the block. This is done in the following way:

Consensus

The block producers exchange messages with the shards for which they have their parts of the chunks. Assuming for simplicity that no block producer proposed multiple chunks (which is a slashable behavior), a block proposal is a bitmask of the shards for which the chunks to be included. A block proposal is only possible if there are 2f+1 block producers that indicated that they have the parts for such chunks. The reward for the block is quadratic in the number of shards included, and thus the block producers have a certain incentive to wait for more chunks to become available before moving to finalizing the block.

At some point each validator indicates that they wish to proceed to finalizing a block. Once 2f+1 wished to proceed, they collectively use arbitrary BFT consensus, without loss of generality Tendermint, to decide on the block, and collect a BLS signature on it.

State Validity

To ensure the state validity, we introduce a second role, called Fishermen. Say there's on the order of 1,000-10,000 fishermen. At the beginning of each epoch each fisherman is randomly assigned to some number of shards, say it is again 4. To get the assignment the fisherman runs a VRF at the beginning of the epoch to randomly sample exactly four shards.

Once the epoch starts, whenever a block is created, the fisherman downloads all the chunks for the shards they are assigned to (conveniently since the chunks are distributed in a form of erasure-coded parts, chunk fetching can be parallelized), and validates them. The fisherman also randomly selects one erasure-coded part for each shard they are not assigned to and fetches it as well.

Once chunks are downloaded and reconstructed, the fisherman validates all the chunks, and creates a message which is either "YAY" if all the parts for all the shards were successfully fetched, and all the chunks in the assigned shards were valid, or "NAY" if at least one action was not successful, with a proof (see about proofs below). The fisherman doesn't reveal the message immediately, instead it commits to it within 10 blocks after the block being validated was produced, and reveals the message within 10 blocks after that.

[

1266×708

](https://lh6.googleusercontent.com/qBJCnoCQjm5FAfoXljJEmt8xzLa0dMw0v0mvrgMWsH2-58MYn2zFvSoRRdC70zjkB22f0CKHXD-C1uZaiC-mMl6yuEaahZ_9kiXTPcB_4QL36tNAR-zCyF1OUwmOftblLjgiXf0_)

A nice property of this fisherman approach is that on the protocol level there's someone who's responsible to fish (so in a sense fisherman here is closer to a concept of a validator), which is different from common approach to fishing in which figuring out incentives for fishermen are not as easy.

The commit reveal scheme makes fishermen accountable. If fisherman A successfully fished, all the fishermen who committed to "YAY" will be slashed (and one must commit to something to get the reward).

Once a fisherman reveals a "NAY" message, they also reveal a proof of the invalid state transition (see below). Every block producer and fisherman needs to validate the proof. If the proof is wrong, the stake of the fisherman who produced it is slashed, and the system continues to operate.

If the proof is correct, every fisherman must reveal their bitmasks within the next 10 blocks, and any fisherman that fails to do so or reveals a bitmask and has the faulty shard assigned to them, but doesn't reveal a corresponding "NAY" message, gets slashed. In this case the block producer and fishermen assignment is immediately redone. However, since it takes time to download the state for all actors involved, the system for some predefined time continues operation with the current assignments, effectively having lower security for some time (higher chance of adaptive corruption, since the fishermen are revealed).

The aftermath of a successful fishing attempt can be modeled as a fork, but we find it easier to think about it as the continuation of the same chain, but with the state transition of all the blocks from the block with the invalid state transition until the block with the successful fishing nullyfied.

Aggregating Fisherman Messages

There are two big problems with 1,000+ fishermen attesting in this way to block validity: a) the blocks will need to include more than 1,000 commitments per block, which is rather expensive, and b) the blocks will either need to include 1,000 cheap

signatures, or one BLS signature, but validating a 1,000-fold BLS signature is relatively expensive.

To get around this issue the following model is proposed: each fisherman is assigned to one block producer. There are no requirements on the assignment procedure (we compare two different models at the end), in particular the fishermen assigned to a particular block producer are not necessarily assigned to the same shards. The role of the block producer is then to perform the following protocol after each block is produced:

1. Collect commits and reveals from each fisherman;
2. Build a merkle tree of such commits and reveals;
3. For each fisherman, send them a merkle root and the merkle proof of inclusion of that fisherman's commits and reveals in the tree;
4. Collect a BLS signature part from each fisherman on the merkle tree root;
5. Include the BLS multisig from assigned fishermen and the merkle root into the block.

Ultimately the block producers choose a sample of other block producers to validate the multisigs from their fishermen, and only sign on the final block if for the sample they verified the multisigs were correct. A particular node that replays the block then can only validate the signature on the block from block producers, and assume that unless a very large percentage of the block producers colluded, the signatures from the fishermen were correct (they can still opt to validate the signatures themselves). Fishermen in consecutive blocks also validate a subset of multisigs of fishermen from the previous block, and if such a signature was faked, within few blocks the majority of fishermen will stop signing the blocks.

A block producer can censor messages from a validator, which we analyze below when we talk about approaches to assigning them. Ultimately the only message that we absolutely cannot lost is a message with a fishing attempt. Such a message is the only message that a fisherman can submit through any block producer. If other messages get censored, the fisherman either chooses to work with another block producer, or waits until the rotation, depending on how they are assigned.

Proofs

Invalid state transition.

A simple way to model it is to force block producers to include a hash of the state after every consecutive segment of transactions that collectively read or write more than 100Mb of state. Then to prove that there was an invalid state transition the fisherman needs to provide two consecutive state hashes, transactions in between, and 100Mb of data that is affected with the Merkle proofs, which is sufficient to prove that the state transition was applied incorrectly.

Data unavailability.

To prove that a particular part of erasure coded state is not available, a participant can send a special kind of request to other participants asking if they can fetch the part. If they can, they return the part, otherwise they return a BLS signature part attesting that they also failed to retrieve the part. A BLS multisig with a large percentage of a stake is a meaningful proof that a part is unavailable.

Fork Choice Rule

Forks shall be unlikely, since the block producers do a BFT consensus on each block. In case a fork occurs, the fork choice rule must favor blocks that have most of the stake from both block producers and fishermen staked on them.

An example fork choice rule that seems to work is to use LMD GHOST that accounts for signatures from both the block producers and fishermen, and slash for signing two conflicting blocks for the same height.

Approaches to assigning Fishermen to Block Producers

We compare two ways to assign fishermen to block producers, and sampling the block producers.

The first approach is for block producers to choose their fishermen, and the 100 block producers with the highest accumulated stake (the stake of the block producer and all the assigned fishermen) will be the block producers. Such block producers naturally won't rotate frequently, which creates certain centralization concerns. However, since to become one of the hundred one needs to sign up participants with a lot of accumulated value, it seems reasonable to assume that any honest fisherman with sufficient stake shall be able to convince at least one of the 100 current block producers to include them.

The second approach is for block producers and fishermen to be sampled from a large set of participants wishing to participate in network maintenance, and assign fishermen to block producers randomly.

In the former approach if a block producer doesn't include fisherman's messages, or the link between them is slow and the fisherman delays the block production, both the block producer and the fisherman can in their sole discretion break their relationship, and the fisherman can go to another block producer. Since the block producer gets a percentage of rewards from their fishermen, this shall end up in an equilibrium in which each block producer ensured high uptime and good link to their fishermen.

In the latter this property will not hold, but the assignment of fishermen to block producers can be reshuffled relatively frequently, and thus censoring fishermen shall not be a big problem.

What's good about it

This approach has several positive properties:

1. Adaptive corruption is extremely complex. To carry out an invalid state transition, the adversary needs to identify the fishermen assigned to a particular shard (for which the only feasible approach appears to be to provide them provable payout exceeding their stakes for YAY-ing an invalid block on another blockchain), and succeed in corrupting each and every one of them, which is significantly more unlikely than corrupting a subset of the validators.

2. For as long as 2f+1 block producers are not intentionally attempting to stall a shard, the data availability story is very good.

3. For as long as at least one fisherman performs their duties, the cross-shard transactions are atomic (in the sense that if an invalid state transition is detected, or the main chain reorgs, it will never be the case that only half of a cross-shard transaction is finalized).

Feedback and brainstorming are highly appreciated!