Tinydancer: Diet Client v0 for the Solana Blockchain

Harsh Patel, Anoushk Kharangate

Abstract

The Solana blockchain has brought about a new paradigm of blockchain scalability with innovations like Proof of History, Turbine and Gulf Stream which have significantly improved TPS and increased activity in decentralised finance, digital asset marketplaces etc. However, the primary trade off for all of these advancements is that running a Solana full node is quite resource intensive and expensive, making it extremely inaccessible for the majority of the users. This can be an attack vector where the corrupt super majority could manipulate blocks and create an invalid state transition which could lead to a safety failure that only full nodes are immune to but not the layman. Anyone should be able to spin up a diet client on a mobile device and verify the validity of their transaction and state without requiring a full node or having to trust the super majority. This paper proposes a solution to this problem by using a combination of Data Availability Sampling and Merkle Proofs to verify the validity of a transaction and state without requiring a full node.

1 Shreds

Shred is a data structure which can be referred to as the smallest unit of a block that are sent across the network in the form of UDP packets using Turbine Block Propagation. Shredding is the process done after creating Entries into the ledger. Shredding is done for two reasons, one being that sending an entire entry at once over UDP is not possible given the IPV4 MTU of 1500 bytes and two shreds are erasure coded so they also help with packet loss and malicious dropping of packets.

Each shred has a sub-type - coding shred and data shred. Data shreds contain the ledger entries. Coding shreds are required to provide redundancy to protect against the network dropping packets. Since there is a chance for UDP networks to drop packets as in contrast to TCP where dropped packets are retransmitted, the need for having erasure coded shreds encoding the data shreds itself is critical due to packet loss. The coding shreds also prevent any malicious node from dropping packets while propagating the shreds.

Each Shred also has the following data layout:

1. Common Header
2. Data or Coding Header
3. Payload

Common Header

The common header contains the signature of the leader which is a signed message originated from the merkle root of the shreds. This is used to validate that the shreds are received from the leader and to slash the leader if the proofs don't compute to the root. It also contains the slot number, the shred version and the FEC index which helps with creating the coding shreds.

1.1 Data Shred

Data shreds are the individually transmitted units of the ledger entries. The payload here contains the path of the shred to the merkle root i.e the proof and the actual merkle root. This allows for doing equality checks without having to reconstruct the whole block. It is important to note that this a newly implemented method and the legacy method didn't have merkle proofs in shreds.

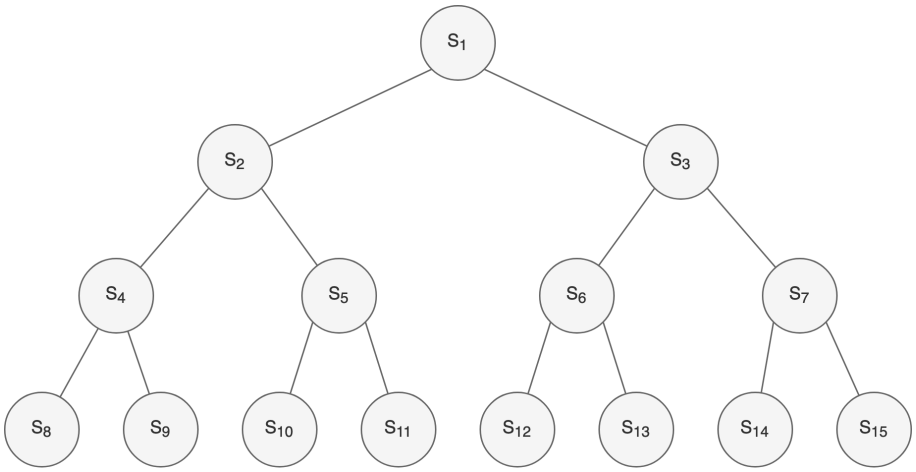

Figure 1: A Merkle Tree with data and coding shreds as leaves. Given S_{10} shred can be validated using proof $\{S_{11},S_4,S_3\}$

1.2 Coding Shred

Coding shreds store the erasure-coded data. Solana uses Reed-Solomon erasure coding, a polynomial encoding algorithm. Given n shreds and k data shreds we get the number of coding shreds t = n - k. The coded shreds are generated using a generator polynomial g(x) = (x-a)(x-a^2)(x-a^3)$\ldots\ldots$(x-a^{2t})

2 Consensus Checks

2.1 Block Headers

Every block has a hash that the staked validators are voting on. The diet client would request the block header from the honest super minority node to get the following information:

1. Vote signature of each validator
2. The identity of the validator
3. The staked amount

Currently Solana doesn't have this concept of block headers so either the RPC client would have to be modified or a better solution since a lot of validators lock their RPC port would be to open a channel specifically for the diet clients who are identified in the diet table similar to gossip table.

2.2 Super majority

Solana blockchain employs PoS or Proof-of-Stake and uses variant of PBFT for consensus called Tower BFT. The network reaches super majority[1] when 2/3 of the staked validators vote on a common fork of the ledger. The block is hence considered to be confirmed and the network has reached a consensus. For light clients it makes sense to use a confirmed block to check for an invalid state transition

Using the headers and the staked validators in the epoch we can check if a certain block has reached super majority consensus.

3 Data Availability Sampling

3.1 The data availability problem

Any blockchain node maintains state or a copy of the ledger that needs to be available to any other node in the network or a client interacting with the network. Assume a probable situation where in the supermajority of the nodes in the network are corrupt and produce invalid state transitions. In addition to these there are a small fraction of nodes who are honest. Now these corrupt nodes can readily censor or 'hide' the invalid state transitions from the honest nodes by withholding the data. This attack is called a data withholding attack. This problem can be mitigated through a mechanism called Data Availability sampling first described in [3]. Data availability can be guaranteed through sampling of block data through various mechanisms as described in this paper [2]. Solana validator clients' Turbine block propagation already achieves the chunking of blocks and erasure coding through a process called shredding. Each shred can be randomly sampled from m random validators which we explore in depth in the section below.

3.2 Implementation

The diet client would request r random samples of shreds from a particular validator from the gossip table. As mentioned before the probability of data being missing is 2^{-r} if 2 is the erasure ratio. Hence r would have to be a value small enough that it's efficient to sample but large enough to reduce the probability of an error. We can calculate it by using this formula

$$\frac{1}{2^r}$$

which would give us the probability of data being withheld given the shreds we sampled. Notice how at 10 shreds the probability is low but then just doubling it to 20 it becomes exponentially lower and we can easily validate around 256 shreds as blocks are much larger.

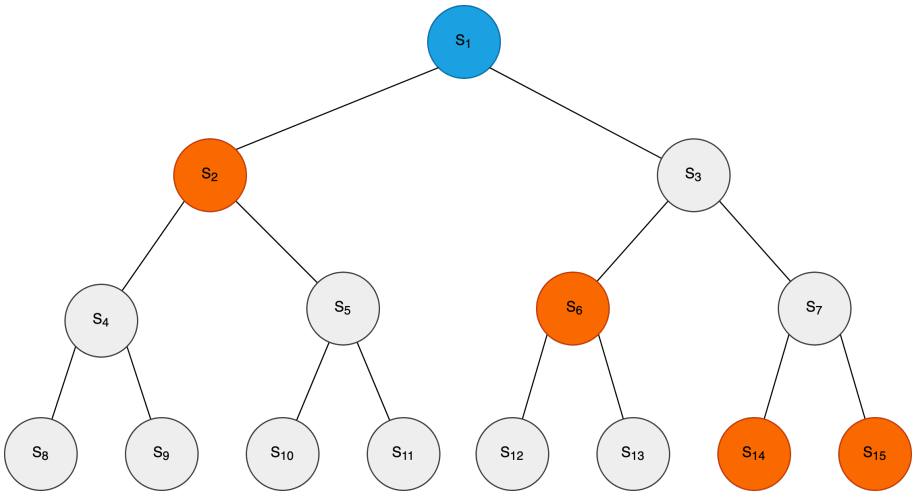

Figure 2: If we want to validate S_{15} we can use proof $\{S_{11},S_4,S_3\}$ which should compute to S_1 if the shred is valid

The validators would have to recompute the shreds from the block in the bank on request from the diet client. Once the shred samples are obtained the diet client will individually check if the proof in the payload computes to the root in the same for each. It will also check the shred headers if the signature is a signed message that contains the root signed by the leader.

4 Protocol Design

4.1 Interactive

For every block processed by the network, the diet clients query the block headers which should contain 2/3 of the staked votes of the nodes in the network, indicating consensus and block confirmation. In addition to receiving the list of validators who voted on the invalid block, clients would also get their stake amounts adding up to super majority stake and the signatures.

Since we don't want diet clients to store the entire ledger unlike a full node they need access to data from the rest of the network. We assume the network has a single honest full node and any light client is at the very least connected to that single honest full node. The diet clients would use DA Sampling for every block as described in the above section. If the client receives all the requested shreds from the validator and it passes merkle authentication then the client can move on to the next stage. If not then it will proceed to request other diet clients in the network recurrently for the missing shreds. Upon failure of these the diet client would know that it should not trust the confirmation and data has been either withheld and/or tampered with.

4.2 Non-Interactive

The above protocol can be made non-interactive by embedding a deterministic randomness function inside the validator client that takes in the Slot number, the diet client's ed25519 public key as seed and the slot hash to generate a random vector to sample the shreds. This would eliminate the need for the diet client to request the nodes for shreds as the nodes would directly send the shreds to that particular diet client based on the corresponding sampled randomness.

5 Gossip Network and Partition Repair

5.1 Cluster Replicated Data Store for Diet Clients

Individually the diet clients would be sampling shreds and storing them for the epoch which would help prevent eclipse attacks as explained above. In addition to that, it also helps the lagging clients to sync up very quickly with others in the network. But the probability of data being withheld exponentially decreases once you have multiple diet clients which communicate with each other about the state of data availability for a particular slot. When a client starts sampling a slot and doesn't receive the necessary shreds it may request other clients in the network to check if the shred has been propagated to any other client. This ensures that there are no false alarms about an eclipse attack that might be caused by some kind of software or network error with that particular client and node. Upon verify the merkle root if there seems to be an invalid shred that too can be confirmed with other clients in the network.

5.2 Client Aided Partition Repair

Validators tend to have to repair their forks either due to inefficiency in turbine or network difficulties. This requires them to contact the leader which can make the repair process slower. Additionally, nodes can be eclipsed from the shreds necessary to repair the partition if the super majority has propagated a malicious block in which case the node cannot repair and replay the block to prove this. To combat this the validator can look at the gossip table and request the shreds that are missing from it's fork from each client and then clients can then send the shreds to the validator. Because partitions can be repaired from diet clients spread across the globe that may be geographically close to the node, the repair process will also be faster by a significant magnitude compared to the from another validator.

References

[1] Anatoly Yakovenko *Solana: A new architecture for a high performance blockchain v0.8.13* https://solana.com/solana-whitepaper.pdf

[2] Paradigm Research *Data Availability Sampling: From Basics to Open Problems* https://www.paradigm.xyz/2022/08/das

[3] Mustafa et al., 2019 *Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities* https://arxiv.org/pdf/1809.09044.pdf