Sszb: A High Performance SSZ Implementation in Rust

Ghilia Weldesselasie
Fellow in the 5th EPF cohort

Ethereum Protocol Fellowship

Agenda

What is Serialization?

Serialization is the process of transforming a data structure into a format (usually an array of bytes) that can be stored or transmitted over a network.

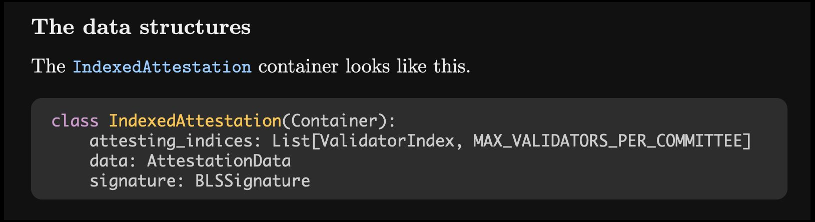
Data structures can't be transmitted as is, they must be serialized on one end and deserialized on the other. This is especially true as their in-memory representation may not be the same across different implementations. This necessitates a common encoding format.

What is SSZ?

Simple Serialize (SSZ) is a new serialization scheme on Ethereum 2, meant to replace Recursive Length Prefix (RLP) encoding on Ethereum 1.

It is not self-describing and uses schemas to determine how to serialize/deserialize data. Schemas are a must have in a performant serialization format. It also allows for some level of direct access.

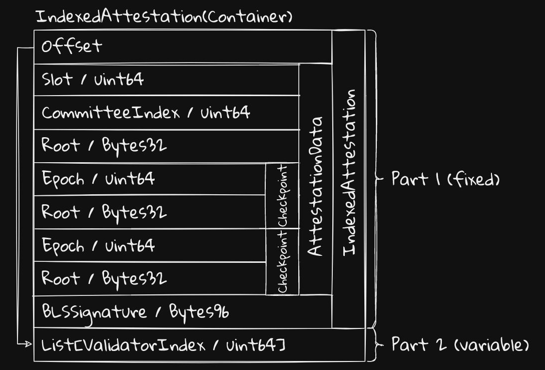
It is deterministic like RLP, so the same data always results in the same serialized output. This is crucial for consensus.



The data structures

The `IndexedAttestation` container looks like this.

```
class IndexedAttestation(Container):
    attesting_indices: List[ValidatorIndex, MAX_VALIDATORS_PER_COMMITTEE]
    data: AttestationData
    signature: BLSSignature
```



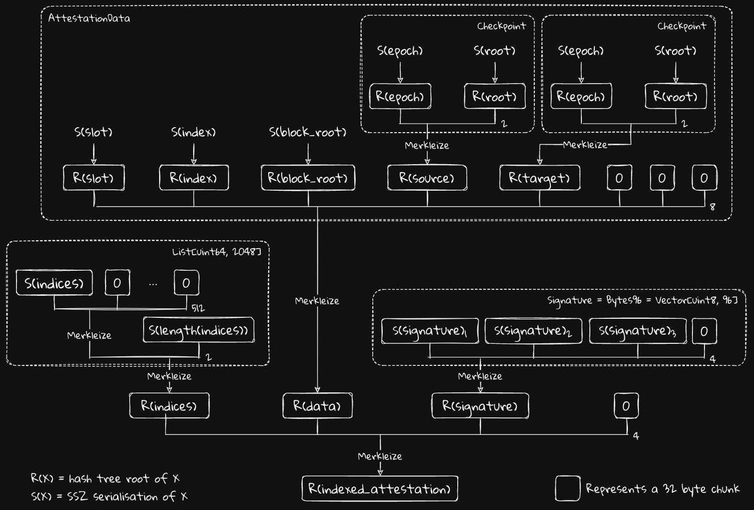
Serialisation of the ***IndexedAttestation*** *container.*

What is Merkleization?

Merkleization and Tree Hashing is the process by which we produce a short digest of state on Ethereum, while allowing for updates without rehashing the *entire state*.

Along with SSZ, this method for constructing root hashes of complex SSZ containers was devised for consistent hashing across the ecosystem.

This is useful for efficient consensus operations, as well as small proofs for light clients that don't want to hold too much state data.



Illustrating the steps required to calculate the hash tree root of an ***IndexedAttestation****. The small digits are the number of leaves in each Merkleization operation.*

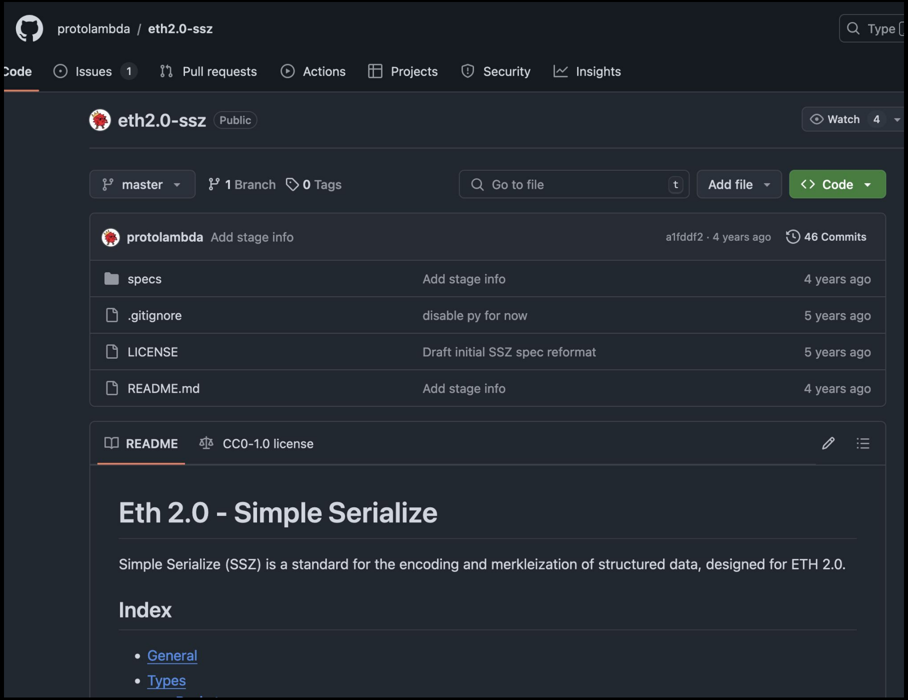
The SSZ Ecosystem

There are a few SSZ implementations in several languages (mostly Go).

Rust has:

- Sigma Prime's `ethereum_ssz`
- Grandine's crate (not public)
- Alex Stoke's `ssz-rs`

For the first half of the fellowship, I worked on a benchmarking suite to to evaluate how these libraries perform against each other.



protolambda Add stage info a1fddf2 · 4 years ago 46 Commits

specs	Add stage info	4 years ago
.gitignore	disable py for now	5 years ago
LICENSE	Draft initial SSZ spec reformat	5 years ago
README.md	Add stage info	4 years ago

README CC0-1.0 license

Eth 2.0 - Simple Serialize

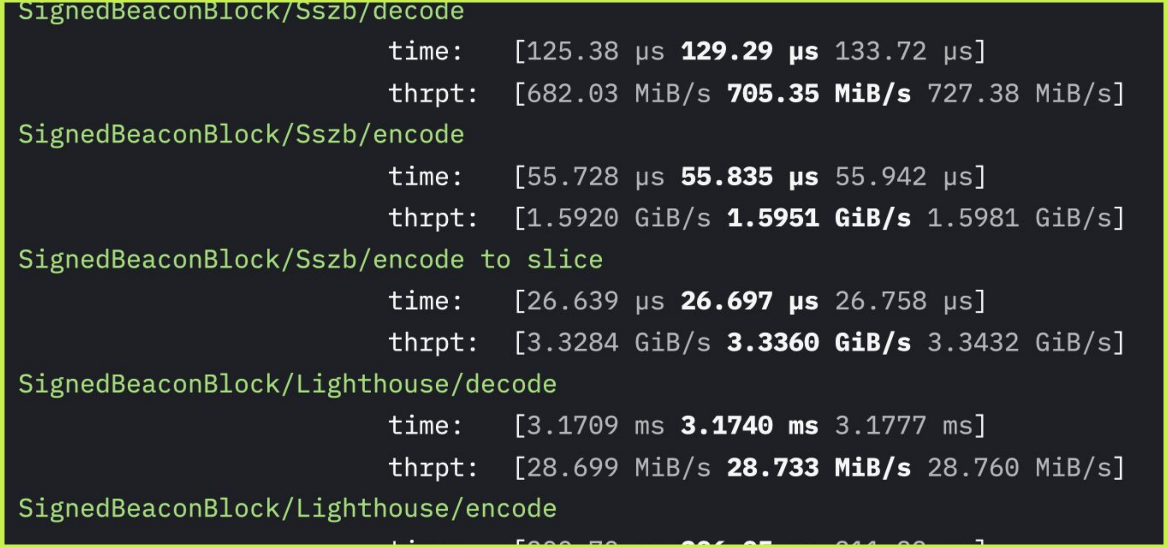
Simple Serialize (SSZ) is a standard for the encoding and merkleization of structured data, designed for ETH 2.0.

Index

- General
- Types

Benchmarks with Ssz-Arena

ghiliweld/ssz-arena



```
SignedBeaconBlock/Sszb/decode
                        time:   [125.38 µs 129.29 µs 133.72 µs]
                        thrpt:  [682.03 MiB/s 705.35 MiB/s 727.38 MiB/s]
SignedBeaconBlock/Sszb/encode
                        time:   [55.728 µs 55.835 µs 55.942 µs]
                        thrpt:  [1.5920 GiB/s 1.5951 GiB/s 1.5981 GiB/s]
SignedBeaconBlock/Sszb/encode to slice
                        time:   [26.639 µs 26.697 µs 26.758 µs]
                        thrpt:  [3.3284 GiB/s 3.3360 GiB/s 3.3432 GiB/s]
SignedBeaconBlock/Lighthouse/decode
                        time:   [3.1709 ms 3.1740 ms 3.1777 ms]
                        thrpt:  [28.699 MiB/s 28.733 MiB/s 28.760 MiB/s]
SignedBeaconBlock/Lighthouse/encode
                        [illegible]
```

LEVERAGING CRITERION FOR USEFUL BENCHMARK REPORTS

ssz-arena is a benchmarking suite that evaluates ssz implementations on test cases and real blockchain data (from a beacon chain checkpoint). This gives a robust way to compare performance.

So How Does One Optimize SSZ?

The boundary between serialization and data structure optimization is blurry. In most cases, the best optimization you can do is optimizing the underlying data structure you're encoding/decoding and making it more amenable to those tasks.

That’s only possible with control over the data structures used. Often, a slower data structure might have some other benefits outside the context of serialization.

Being constrained by the inability to change the underlying data structure, I opted to minimize intermediate allocations.

Sszb Design Decisions

ghiliweld/sszb

BUF AND BUFMUT TRAITS

Gives developers the flexibility to use Vecs or slices or any buffer type that implements the Buf/BufMut traits. It also has the added benefit of abstracting the offset accounting, greatly simplifying the implementation at zero cost.

ALLOCATION MINIMIZED

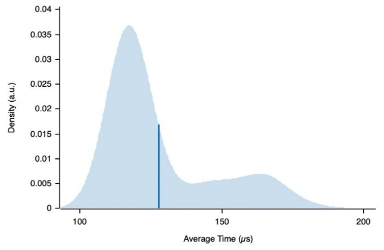
We avoid intermediate state during the encoding process, and minimize the intermediate state needed during decoding. This greatly reduces the number and size of the memory allocations needed to perform serialization, which is the dominant cost.

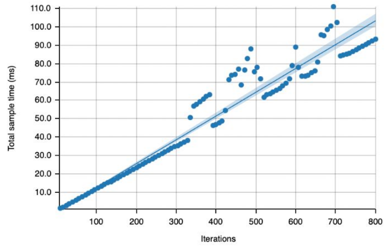
Peter’s Go implementation and Grandine’s were big inspirations.

Plotting Performance of Beacon Block Decoding

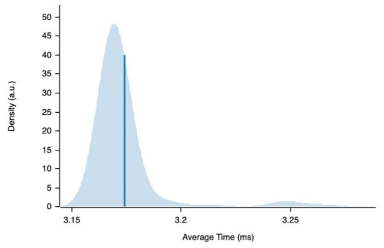
measurement of the given function/parameter would take a particular length of time.

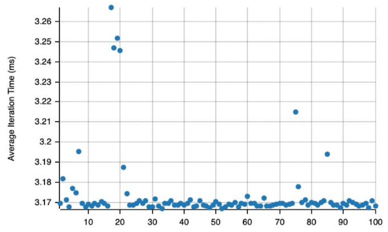
SignedBeaconBlock/Sszb/decode





SignedBeaconBlock/Lighthouse/decode





~129 µs

sszb decoding

~3 ms

ethereum_ssz decoding

How Sszb Performs on Beacon Blocks (by the Numbers)

Faster Encoding..

ENCODING SPEEDUP

~85%

and Faster Decoding

DECODING SPEEDUP

~95%

Next Steps

SszHash

Shipping support for merkleization and merkle proofs with generalized indices.

SszCheck

Support input validation that some input bytes conforms to a type T encoding to reject malformed inputs earlier.

Stable Release

Gear up for a stable release, adding usage docs as well as doc comments before publishing.

Partial Encoding & Decoding

Supporting decoding subsets of an input. For large objects like beacon state, full decoding is rather expensive so partial decoding would be a game changer. Re-encoding and re-hashing work similarly for sparse updates.

Q&A

Ghilia Weldesselasie

ghiliaweld@gmail.com

@ghiliweld on Twitter

Mentor

Michael Sproul @
Sigma Prime

@sproulM_

Sources

- https://eth2book.info/capella/part2/building_blocks/ssz/
- https://eth2book.info/capella/part2/building_blocks/merkleization/
- https://epf.wiki/#/wiki/CL/SSZ
- https://epf.wiki/#/wiki/CL/merkleization
- https://github.com/ghiliweld/ssz-arena
- https://github.com/protolambda/eth2-docs
- https://github.com/ghiliweld/sszb
-