It’s time to start laying out how best to design light clients for the current Casper beacon chain spec (and the Casper FFG family more generally). There are a few major challenges at play:

- The validator set is potentially quite large, and there are a lot of signatures to verify
- In the original Casper FFG spec, the validator set could change arbitrarily within 2 dynasties, which could take place within 40 minutes, meaning that there was no way a light client could quickly “skip through” the chain, as verifying epoch N would give them no guarantees about what signatures to trust to validate epoch N+4
- Verifying validator set changes would require Merkle proofs, which leads to an additional O(log(N)) factor overhead.

I’ll write out the numbers for old-style Casper FFG, though assuming 32 ETH deposit sizes to make a fair comparison. Suppose 10 million ETH is participating, so there are 312500 validators. Each epoch would require signatures from all of these validators: 312500 * 100 (64 plus overhead) = 31.2 MB. Each dynasty would require checking the new validator set. Because the validator set was saved in a contract along with a lot of other data, this might be 312500 entries in a trie with, say, 100 million keys. The size of this is roughly 128 * (log(100 million) - log(312500)) * 312500 + 32 * 312500 = 227 MB

. This data would have to be downloaded and checked every 20 minutes

, or 107 KB/sec. For a light client

.

Now, let’s start making optimizations from this. First of all, we store the validator set in a separate tree, and make it a simple Merkle tree. Now the data complexity of checking the new validator set is just 32 * 312500 = 10 MB. Next, we replace the 64-byte signatures with BLS aggregate signatures; suppose we follow the 2.1 spec, which would mean we have 1280 BLS signatures per epoch. The signature data complexity goes down to 1280 * 64 = 81920 bytes, plus a bitfield, of 312500 bits (39062 bytes). So we’re at 121 KB for sigs, so 10.1 MB

total.

We can introduce into the data structure a linked list that we’ll call a “recent change log”, which simply is a list of the form “index 87124 logged in with key 0x7182641b5, index 88029 logged out, index 40125 logged in with key 0xab41265b3…”. Given that logging in or out incurs a delay of 4 months (~=8333 epochs), even if all ETH (assuming 100 million) is logging in and out as fast as possible, that’s only a maximum possible 750 changes per epoch, and more realistically it would be ~1-50. We’ll take a conservative 100. A log-in would be ~36 bytes and a log-out ~4 bytes, so that’s an average of ~20 bytes, so our estimate would be 20 * 100 = 2000 bytes per epoch. If we require light clients to verify proofs-of-possession, add 64 bytes for a log-in, so that’s 5200 bytes per epoch for validator set updates. Yay, 126 KB

!

Now, we can start digging a bit deeper. At this point, a client needs to verify 1280 BLS signatures per epoch. But what if a client is willing to accept a probabilistic assurance? Suppose the client checks two signatures per block: the proposer and a randomly selected one, so that’s a total of 128 per epoch. This represents a randomly selected fraction, so it would be difficult to come up with those signatures without also having the other signatures, so it gets us pretty close. An attacker who does

have over 50% can possibly fool some light clients without getting slashed, but attackers with much less than 50% would have no way of sufficiently filling up even one threshold signature. So now we go down to ~12.1 KB for sigs. We can now also separate the login/logout trace from the proofs of possession, and allow clients to not verify the proofs of possession, trusting the validators to verify those, cutting the validator set trace back down to 2 KB. Yay, 14.1 KB

.

Our last optimization will be to skip epochs, verifying signatures for one in every twelve epochs. A dynasty change can happen at most once per four epochs (in the current spec), and can only change 3% of the validator set, so this would cause the client to check signatures against a validator set that is at most off by 12%. The client would still need to download all validator set change logs to verify the aggregate signatures, but would not need to check all signatures. Now, the signature checking per epoch drops to ~1.01 kB for sigs. And so we’re finally at 3 KB

.

The current PoW chain requires checking 500 bytes per 14 seconds, or ~42.8 KB per 20 minutes, so even without randomly sampling signatures we only get an efficiency drop of ~3x, and with random sampling we get a significant boost in efficiency. If desired, the light client can also ask for the BLS signatures to be sent to them in an aggregated form, saving almost all of the 1 KB, though not the computational overhead. The remaining overhead would then be maintaining the up-to-date validator set.