See Moving ETH between shards: the problem statement for the problem statement and A meta-execution environment for cross-shard ETH transfers for an earlier attempt.

Solution

Every shard stores a map balances: (shard, EE) \rightarrow balance

. The total “real balance” of some EE in some shard can be computed as:

real_balance(shard, ee) = \sum_{i=0}^{shard_count} shard[i].balances[s][x]

To perform any transfers of ETH between EEs in a block, the shard must contain a Merkle proof from the most recent state of every shard, showing shard[i].balances[s][x]

for every shard i

to prove the total balance of the EE. If a transfer between EEs is made on some shard s

, transfering xfer_amount

ETH from EE ee_1

to ee_2

, then we check that the real_balance(s, ee_1) >= xfer_amount

, and then set shard[s].balances[s][ee_1] -= xfer_amount

and shard[s].balances[s][ee_2] += xfer_amount

.

To perform a cross-shard transfer from shard s1

to shard s2

, we set shard[s1].balances[s1][ee] -= xfer_amount

and shard[s1].balances[s2][ee] += xfer_amount

.

Note that invididual shard[s].balances[s][ee]

values may sometimes be negative. For example, if the initial balances for some EE are {A: [1, 0, 0], B: [0, 0, 0], C: [0, 0, 0]}

and then 1 ETH is transferred from shard A to shard B, and then soon transferred from shard B to shard C, the final balances would be {A: [0, 1, 0], B: [0, -1, 1], C: [0, 0, 0]}

. However, the “real balance” \sum_{i=0}^{shard_count} shard[i].balances[s][x]

should always remain non-negative.

Overhead

Under conditions of high usage, about the same overhead will be required as previous schemes (ie. ~20 kB per EE), though slightly lower because Merkle branches into bitfields will not be required. Also, if a block contains exclusively proofs within an EE, or if for every EE, ETH coming in equals ETH going out, proofs are not required. However, if there are unbalanced cross-EE transfers, then full proofs are required for any EE that has net-outgoing funds.

Further improvements

EEs could have a “reserve” of funds saved on every shard (eg. 1 ETH per shard). If a particular EE has more outgoing than incoming transfers in a block, then that reserve would be reduced; full balance proofs would only be required when the reserve reaches zero. If an EE has more incoming than outgoing transfers, then the reserve would be increased.

A “zero capital overhead” alternative would be a system where shard[i].balances[j][ee]

stores both the balance value and also the amount received minus amount spent in the most recent slot. A block in shard j

spending amount X could avoid providing a full proof of all shards for some EE by providing some proofs from some other shards containing proofs of amounts available from the previous slot.