With the proposal for withdrawal methods by@djrtwo it raises an issue that this would result in users being able to do an "exit-withdraw-deposit" loop to move "excess" balance from a validator (excess being anything over MAX_EFFECTIVE_BALANCE). Any significant level of validator churn to serve this purpose would be both annoying for users, as the activation and exit queues would have a baseline level of business, and the network, as it has to both process exits and activations, and retain exited validator state.

An alternative is to allow validators to transfer their excess balance without exiting. Some sort of "transfer" operation would be the obvious way to go, but it could inadvertently creating an arbitrary value transfer system on the beacon chain. Ideally, a way would be found to enable this functionality without requiring additional operations.

The proposal is to allow validators to transfer excess balance at the point they propose a block. This makes the transfer an inherent part of the block proposal, using existing security and limiting the number of transfers that can take place. The only data impact is an additional 8 bytes per beacon block.

Beacon chain changes

- add transfer_index: ValidatorIndex

to BeaconBlock

/ BeaconBlockHeader

Process

- if a validator wishes to transfer its excess balance it sets the beacon block's transfer_index

field to the index of the validator to which it wants to transfer

- as part of process_block_headers

: * if transfer_index

is set then any balance over MAX_EFFECTIVE_BALANCE is transferred from the proposing validator to the validator specified by transfer_index

as long as both of the following conditions are met:

- len(state.validators) > blocks.transfer_index
- state.validators[block.transfer_index].activation_eligibility_epoch == FAR_FUTURE_EPOCH
- if transfer_index

is set then any balance over MAX_EFFECTIVE_BALANCE is transferred from the proposing validator to the validator specified by transfer_index

as long as both of the following conditions are met:

- len(state.validators) > blocks.transfer_index
- state.validators[block.transfer_index].activation_eligibility_epoch == FAR_FUTURE_EPOCH

Wrinkles

Because transfer_index

is a validator index 0

is a valid value, there is not an obvious "no index" value. 0xff..ff

could be used for this purpose, or 0

could continue to be used as validator 0 will not meet the second condition above (at least, for mainnet). Use of a BLSPubKey

for the transfer

field would be an alternative, but that would increase both storage and CPU requirements.