TLDR: the user experience

If you are an application developer or a user, and the roadmap described in this post is used to complete the eth1 -> eth2 transition, the changes and disruptions that you experience will actually be quite limited. Existing applications will keep running with no change. All account balances, contract code and contract storage (this includes ERC20 balances, active CDPs, etc etc) will carry over.

What you will

have to deal with is the following:

1. Gas cost of IO-accessing opcodes (SLOAD, BALANCE, EXT*, CALL*) will increase. Gas cost of CALL will likely add a gas cost something like 1 gas per byte of code accessed.
2. At some point, you will have to download code that implements the network upgrade. This is fundamentally not different from any other upgrade, eg. Byzantium, Constantinople, but it's a little bigger download because you'll need to get an eth2 client if you do not already have one.
3. The chain will likely halt for ~1 hour. After 1 hour, it will look like "Ethereum" is back online, except at that point eth1 would be functioning as a subsystem inside eth2 instead of being a standalone system.

That's it. If you are a developer, you can eliminate the largest part of disruption from gas cost changes by proactively making sure you don't write apps with high witness sizes, ie. measure the total storage slots + contracts + contract code accessed in one transaction and make sure it's not too high.

How the transition may happen

Suppose that phases 0-2 have happened, and the eth2 chain is stably running. The eth1 chain continues to stably run as well. In the phase 0 spec, there already exists a mechanism, [eth1_data

voting](https://github.com/ethereum/eth2.0-specs/blob/fffdb247081b184a0f6c31b52bd35eacf3970021/specs/core/0_beacon-chain.md#eth1-data), in which validators vote to agree on a recent canonical eth1 hash; this mechanism is used to process deposits. We will simply repurpose this mechanism to feed the full state (root) of eth1 into eth2.

Currently, this mechanism has a ~6 hour delay (4 hours from the ETH1_FOLLOW_DISTANCE + 2 hours from the voting period), but these parameters could be reduced over time before the transition to make the delay ~1 hour.

The basic mechanism to effect the transition is as follows:

[

Transition(1)

1088×309 19.5 KB

](https://ethresear.ch/uploads/default/original/2X/c/cb1b0cb34db05b9cb8370d3da9dc623bdd9ed17f.png)

1. Specify a (eth1-side) height TRANSITION_HEIGHT

. The TRANSITION_HEIGHT

'th eth1 block will be considered the "final" block on the eth1 side; from then on, the "canonical" eth1 will be functioning as a subsystem of eth2.

1. In line with (1), add a change to the eth2 "honest validator" code that disallows voting for eth1 blocks with number > TRANSITION_HEIGHT

. If the voting algorithm would have previously selected some block with number > TRANSITION_HEIGHT

, vote for its ancestor at number TRANSITION_HEIGHT

instead.

1. Additionally, in the case were (2) is triggered, validators should set the deposit_count

to 2**63 higher than its true value (this is basically using the top bit of the deposit_count

as a flag saying "eth1 is finished")

1. When the eth2 chain accepts an eth1data

with the "eth1 is finished" flag turned on, it performs a one-time "irregular state change" that puts the post-state root of that eth1 block into the state of an “eth1 execution environment” (a type of system-level smart contract on eth2). An amount of ETH equal to the total supply of ETH on the eth1 side is added to this eth1 EE’s balance.

After this point, the transition is complete. The eth1 chain technically continues but it is valueless; eventually it will die off when the difficulty ice age hits.

The eth1 system now lives inside of eth2; hence, further transitions to the eth1 system happen by submitting a transaction on eth2 which targets the eth1 EE (which as mentioned above is a subsystem of eth2). The eth1 EE has code that implements the entire eth1 EVM and transaction processing logic; it has a function update(state_root, transaction, witness) -> new_state_root

which takes a transaction and witness (Merkle proofs of portions of the state), processes the transaction and determines the updated eth1 state root, according to the same rules as on the eth1 chain. See The Stateless Client Concept for how witnesses and state roots work.

Additional functionality would be added into the eth1 EE code that allows ETH and messages to be withdrawn from the eth1 EE into other parts of eth2, and into copies of the eth1 EE on other shards. By default, all eth1 accounts/contracts would be placed on the same shard, so to take advantage of eth2’s increased capacity you would need to proactively use this functionality to move your ETH or other applications into other shards, but this would not be difficult. An extension to the ERC20 standard would need to be made to support cross-shard transfers of tokens.

How the user client would work

The user-facing side of the client would be modified before the transition to have two code paths. The client would check eth2 to see if the transition has already happened. If it has not yet happened, then it would send transactions, check balances, etc using eth1 as before, except it would pretend that all eth1 blocks with number > TRANSITION_HEIGHT

do not exist. If the transition has happened, it would look into the eth1 EE on eth2. A full client would process all transactions targeting the eth1 EE on eth2 sequentially, so as to continue updating the full eth1 state tree; this would allow the client to generate witnesses for any transaction they want to send and “package” it in the eth2 format. Light clients (as well as wallets such as metamask) would broadcast their transactions to a full client that could add witnesses for them.

From a user’s point of view, ethereum would “feel” the same pre- and post-transition (except that post-transition it would feel smoother due to PoS and EIP 1559). Very different code paths would be used to package and broadcast the transaction, but the functionality provided would be the same.

Potentially, the transition could even be engineered so that wallets that talk to clients via RPC do not need to change anything at all.

Example user story

You have a CDP on MakerDAO. You go to sleep, and when you wake up the transition has happened. You are able to interact with and liquidate your CDP by sending transactions as before, except your client code would see that you are post-transition and add witness data to your transaction and send it to the eth2 network instead of the eth1 network.

Possible optimizations

During the period between the eth1 chain reaching TRANSITION_HEIGHT

and the eth1 EE on eth2 being fed with that state, we could do some preprocessing on the eth1 state. Particularly, we could:

- Replace the hexary Patricia tree with abinary sparse Merkle tree and a specialized hash function to ensure the hash overhead of branches remains O(log(n)). This reduces the size of Merkle branches by ~4x
- Replace RLP with SSZ hash trees
- Add state rent-related data fields to accounts
- Clear out “dust” accounts
- Modify account structure in line with abstraction proposals

Instead of including the actual eth1 state root into the EE, we would include the root of the state tree generated by performing all of these modifications. This is a deterministic calculation, so all validators could do it in parallel. This one-time expenditure of computation could greatly improve the efficiency and usability of eth1 post-transition.