I was doing some research on state channels as to how to implement them and things of-chain.

I ended up with a basicTicTacToe game which is a minimal VM enforcing [tictactoe] rules with state transition evolving with a 2-2 multisignature scheme with a geth based kind-of Oracle.

As a follow-up, I'm startingSolidity-EVM which I intend to use as an EVM (inception!), it's essentialy to have something that would avoid any modification of the current protocol, I'm just laying out ideas on how I might end up with a prototype that can run a Solidity Morpion.

IDEA:

The goal is to have a DApp thread layer started by using a specialy formated LOG(CREATE3/THREAD?), a private (JSON-RPC full duplex client) daemon would listen for that particular LOG on a contract and instanciate a special function that would be run and each (or batch) instruction should be approved by a basic multi-signature scheme as previous.

To close a channel a special function should be transacted on-chain that would settle the channel using the smart contract last BREAKPOINT (which would open a window for challenge).

Partial state change may also occur with either a receipt mecanism or multisig mecanism w/ partial state.

Upon instanciation a new private state should be instantiated in the micro service memory and signed locally.

Clients should use javascript with something likehttps://medium.com/metamask/scaling-web3-with-signtypeddata-91d6efc8b290

The problem will be the high latency due to signing each instruction (or batch).

Dispute mecanism:

Something kind of likeBREAKPOINT where the miner here would be the Oracle (hence the init state would be BREAKPOINT 0) and the fee mecanism to retribute the Oracle operator in the contract.

Limitations:

Even if that would induce a heavy overhead in all kind of ways, this is a toy experimentation.

SmartContract / Solidity:

LOG: CREATE3(ctor_arg, TDB

, salt)

Contract code should be pulled from on-chain code.

LOG: BREAKPOINT(nonce, storage_root, TDB

)

Bare EVM at first no optimisation.

Routine initialization should set initial state with ctor_arg and return new state with an id(hash?).

BREAKPOINT: A special LOG event with args that should hint or specify dispute resolution, nonce is used for the challenge mecanism where the highest nonce beat the lowest.

Function: Close/Settlement/Challenge that takes BREAKPOINT messages + signatures + storage and do the settle according to the function, an helper function should be provided for verification.

If possible off-chain, the BREAKPOINT should serve as a recovery mecanism to pursue the use of the channel up to last BREAKPOINT in case of trouble.

Program Counter update consensus mecanism:

const uint _ThreadHash = 0; // This is changed to something else based on CREATE3 args when run by the Oracle

function ThreadRun(**TBD**) { while (true) { // do things // Local state is updated in each EVM BREAKPOINT(challengeMecanism / *function* /, multisig / *struct to use w/ previous func.*/, **TBD**) // A receipt or kind-of is produced that can be used w/ resulting (partial or final) state on-chain that might change on-chain state // Return values are pushed onto the stack and retrieved through JULIA } }

Javascript browser-side:

- Use web3 1.0 for websocket streaming purpose

- Should check on BREAKPOINT(/TRAP?) for user interaction
- Run the evm, sign state, publish message in channel

Golang micro-service:

- geth client
- Service Oracle
- Run the evm, sign state if needed, maintain channel active

Frameworks:

- Docker
- Golang
- Javascript

Main libraries:

- web3 1.0 because websocket!
- go-ethereum
- go-swagger / OpenAPI ?

Keep an eye on:

- https://github.com/ethereum/EIPs/pull/712 (eth_signTypedData)
- https://github.com/ethereum/EIPs/issues/719 (Trustless signing protocol)
- https://github.com/ethereum/EIPs/issues/726 (Eval)