In an effort to implement the functionality described in the post State-minimised executions , I've worked up the following contract for a 'virtual token' that allows for the holding of state in a Double-batched Merkle log accumulator as proposed by @JustinDrake.

This is a toy implementation to show what I mean by 'virtual functions'. The idea being that users would call the Transfer() function to emit an event that indicates that they want to transfer. The Collators respond to that call bo collecting witnesses from the provided dataHash(perhaps via IPFS) and using the vTransfer() function to calculate the logs.

My original thought was to have the virtual function emit logs as it was run in an off chain evm, but with current tooling those logs are discarded or ignored so instead accumulate the new logs in the results bytes. This makes the code messier, but it generally works.

```
pragma solidity ^0.4.18;

import "./BytesLib.sol";

contract VirtualToken {

address public owner; uint256 public totalSupply;

event NewBranch(bytes32 indexed addressFrom, bytes32 amountFrom, bytes32 indexed addressTo, bytes32 amountTo, bytes32 indexed parent);
event SwapLeaf(bytes32 indexed oldLeaf, bytes32 indexed newLeaf);

function VirtualToken() public{ owner = msg.sender; totalSupply = 1000000 * 10**18; DataLog(keccak256(address(this), "balance", msg.sender),1,0, bytes32(totalSupply)); }

function getKeyProof(bytes32 key, bytes data) pure returns(bytes32[] result){ result = new bytes32; uint bytePosition = 0; while(bytePosition <= data.length){ uint length = uint256(BytesLib.toBytes32(BytesLib.slice(data, bytePosition, 32))); bytes32 thisKey = BytesLib.toBytes32(BytesLib.slice(data, bytePosition + 32, 32)); if(thisKey == key){ for(uint thisGroup = 1; thisGroup < length; thisGroup++){ bytes32 aGroup = BytesLib.toBytes32(BytesLib.slice(data,bytePosition + (32 * thisGroup), 32));

            result[thisGroup - 1] = aGroup;
        }
        return ;
    }
    bytePosition = bytePosition + (32 * length);
}

}

event DataLog(bytes32 path, uint logType, uint nonce, bytes32 value);

bytes32[8192] public bottomLayer; uint bottomLayerPosition; bytes32[] public topLayer;

function pushLog(uint logType, uint nonce, bytes32 value, bytes32[] proof, bytes startBytes) internal returns(bytes result) { result = startBytes; result = BytesLib.concat(result, BytesLib.fromBytes32(proof[0])); result = BytesLib.concat(result, BytesLib.fromBytes32(bytes32(logType))); result = BytesLib.concat(result, BytesLib.fromBytes32(bytes32(nonce))); result = BytesLib.concat(result, BytesLib.fromBytes32(value)); //DataLog(proof[0], logType, uint(proof[2]) + 1, value); }

function pushDel(bytes32[] proof, bytes startBytes) internal returns(bytes result){ result = startBytes; result = pushLog(2, uint(proof[2]), proof[3], proof, result); }

function pushAdd(bytes32 newValue, bytes32[] proof, bytes startBytes) internal returns(bytes result){ result = startBytes; result = pushLog(1, uint(proof[2]) + 1, newValue, proof, result); }

function publishCollation(bytes32 newBottomCollation, bytes32 newTopCollation) public { if(newTopCollation == 0x0){ bottomLayer[bottomLayerPosition] = newBottomCollation; bottomLayerPosition++; } else { topLayer.push(newTopCollation); bottomLayerPosition = 0; }

}

function vTransfer(bytes data) view returns(bytes results){

address sender = address(getKeyProof(keccak256("msg.sender"), data)[1]);
bytes32[] memory senderProof = getKeyProof(keccak256(address(this), "balance", sender), data);
require(verifyProof(keccak256(address(this), "balance", sender), senderProof));
uint senderBalance =  uint(senderProof[3]);

address destination = address(getKeyProof(keccak256("destination"), data)[1]);
bytes32[] memory destinationProof = getKeyProof(keccak256(address(this),"balance", destination), data);
require(verifyProof(keccak256(address(this), "balance", destination), destinationProof));
uint destinationBalance = uint(destinationProof[3]);

uint amount = uint(getKeyProof(keccak256("amount"), data)[1]);

require(senderBalance >= amount);

//invalidate existing proofs
//todo: what if an item goes to 0

results = pushDel(senderProof, results);
if (destinationBalance > 0) {
 results = pushDel(destinationProof, results);
}

//publish new proofs
destinationBalance = destinationBalance + amount;
senderBalance = senderBalance - amount;

if(senderBalance > 0){
    results = pushAdd(bytes32(senderBalance), senderProof, results);
}

if(destinationBalance > 0){
    results = pushAdd(bytes32(destinationBalance), destinationProof, results);
```

```
}

return results;

}

/* things that need to be in the log inputDataHash: -> maybe put everything but signature here sender: gasPrice: maxGas: timeOut: contract: function: data: value: nonce: signatureInputDataHash: -> to ecrecover sender

//value will need to be stored in escrow until the op can be proven to have run and a reciept generated

//structure of data passed to a function

[length][variableName][loghash][value][map][map][map][map][proof]...[proof]

[length][path][nonce][type][value][proff]...[proof]

*/

event Transfer(bytes32 dataHash, bytes signature);

function transfer(bytes32 dataHash, bytes sig) public{ Transfer(dataHash, sig); }

//utility function that can verify merkel proofs //todo: can be optimized //todo: move to library function calcRoot(bytes32 path, bytes32[] proof) constant public returns(bytes32 lastHash, uint logType, uint nonce, bytes32 proofPath, bytes32 value){ for(uint thisProof = 0; thisProof < proof.length; thisProof++){ if(thisProof == 0){ //path require(path == proof[thisProof]); proofPath = path; } else if(thisProof == 2){ nonce = uint(proof[thisProof]); } else if(thisProof == 1){ //type logType = uint(proof[thisProof]); if(logType == 3){ //null return; } } else if(thisProof == 3){ value = proof[thisProof]; lastHash = keccak256(path,logType, nonce, value); } else if(proof[thisProof] == 0x0){ return; } else{ if(proof[thisProof] == lastHash){ if(proof[thisProof + 1] != 0x0) { lastHash = keccak256(lastHash, proof[thisProof + 1]); } else { lastHash = keccak256(lastHash); }

    thisProof++;
  } else {
    require(proof[thisProof + 1] == lastHash);
    lastHash = keccak256(proof[thisProof], lastHash);
    thisProof++;
  }
 }
}
return;

}

//utility function that can verify merkel proofs //todo: can be optimized //todo: move to library function verifyProof(bytes32 path, bytes32[] proof) constant public returns(bool){ bytes32 lastHash; bytes32 emptyBytes; bytes32 value; uint nonce; uint logType;

(lastHash, logType, nonce, path, value) = calcRoot(path, proof);

if(nonce == 3){ return true; }

for(uint thisLayer; thisLayer < bottomLayer.length; thisLayer++){ if(bottomLayer[thisLayer] == lastHash){ return true; } }

for(thisLayer = 0; thisLayer < topLayer.length; thisLayer++){ if(topLayer[thisLayer] == lastHash){ return true; } } return false; }

}
```

The following code builds a little tree generator that helps manage the state of a tree and to produce Merkle proofs. These proofs can probably be streamlined as I'm including derived hashes in the proofs for simplicity's sake. Although since we are doing these virtually the size of the proofs only affects the bandwidth for collators, and the proofs aren't that big.

```
class DataLogTree constructor: ()-> @web3Utils = require('web3-utils') @root = null @layers = [] @layers.push([]) addItem: (logType, nonce, path, value)=> @layers[0].push [logType, nonce, path, value] getVar: (key, value)=> map = [] map.push(@web3Utils.padLeft(@web3Utils.toHex(3),64)) map.push(key) map.push(value) return map getNullProof: (key)=> map = [] map.push(@web3Utils.padLeft(5,64)) map.push key map.push(@web3Utils.padLeft(@web3Utils.toHex(3),64)) map.push(@web3Utils.padLeft(@web3Utils.toHex(0),64)) map.push(@web3Utils.padLeft(@web3Utils.toHex(0),64)) return map proofBytes: (items)=> bytes = "0x" items.map (item)=>

  item.map (subItem)=>
    bytes = bytes + subItem.slice(2)
    return
  return
return bytes

proofBytesArray: (myBytes)=> @web3Utils.hexToBytes(myBytes) parseLogs: (logsBytes)=> logsBytes = logsBytes.slice(2) position = 0 logs = [] while position < logsBytes.length logs.push [ "0x" + logsBytes.substring(position,position + 64) "0x" + logsBytes.substring(position + 64,position + 128) "0x" + logsBytes.substring(position + 128,position + 192) "0x" + logsBytes.substring(position + 192,position + 256) ] position = position + 256 return logs getProof: (type, nonce, path)=> map = [] seek = null console.log type console.log nonce console.log path # 'looking for ' + key # to produce a proof we look through each layer from bottom to top looking for first the # passed in key and then the hash combination for thisLayer in [0...@layers.length] console.log 'seeking layer '+ thisLayer for thisItem in [0...@layers[thisLayer].length] console.log 'inspecting:' + thisItem console.log @layers[thisLayer][thisItem] if thisLayer is 0 console.log 'in 0' if @layers[thisLayer][thisItem][0] is path and @layers[thisLayer][thisItem][1] is type and @layers[thisLayer][thisItem][2] is nonce console.log 'found 0' map.push @layers[thisLayer][thisItem][0] map.push @layers[thisLayer][thisItem][1] map.push @layers[thisLayer][thisItem][2] map.push @layers[thisLayer][thisItem][3] console.log map seek = @hasher map[0], map[1], map[2], map[3] console.log 'new seek is ' + seek break else # The found item will be either on the left or right hand side if @layers[thisLayer][thisItem][0] is seek # console.log 'found seek in position 0' # push the item onto the proof and find the next item map.push seek if @layers[thisLayer][thisItem][1]? map.push @layers[thisLayer][thisItem][1] seek = @hasher @layers[thisLayer][thisItem][0], @layers[thisLayer][thisItem][1] else map.push @web3Utils.padLeft(0,64) seek = @hasher @layers[thisLayer][thisItem][0] console.log 'new seek is ' + seek console.log map break if @layers[thisLayer][thisItem][1] is seek #console.log 'found seek in position 1' # push the item onto the proof and find the next item map.push @layers[thisLayer][thisItem][0] map.push seek seek = @hasher @layers[thisLayer][thisItem][0], @layers[thisLayer][thisItem][1] console.log 'new seek is ' + seek console.log map break if thisItem is @layers[thisLayer].length throw 'seek not found' if seek is @root map.unshift(@web3Utils.padLeft(@web3Utils.toHex(map.length + 1),64)) return map else throw 'root not found' hasher:(val1, val2, val3, val4) => if val4? hash = @web3Utils.soliditySha3({t:"bytes",v:val1},{t:"bytes",v:val2},{t:"bytes",v:val3},{t:"bytes",v:val4}) else if val2? hash = @web3Utils.soliditySha3({t:"bytes",v:val1},{t:"bytes",v:val2}) else hash = @web3Utils.soliditySha3({t:"bytes",v:val1}) return hash buildTree: ()=> if
```

```
@layers[1]?.length > 0 @layers = [@layers[0]] console.log @layers[0].length pair = [] currentLayer = 0 console.log 'currentLayer:' + currentLayer console.log 'currentLayer Length:' + @layers[currentLayer].length

if @layers[currentLayer].length is 1
  console.log @layers[currentLayer]
  hash = @hasher @layers[currentLayer][0][0], @layers[currentLayer][0][1], @layers[currentLayer][0][2], @layers[currentLayer][0][3]
  @root = hash
  console.log @root
  return

while @layers[currentLayer]? and @layers[currentLayer].length > 1
  console.log 'in layer loop'
  console.log currentLayer
  console.log @layers[currentLayer]
  console.log 'end'

  @layers.push []
  console.log @layers
  #console.log @layers[currentLayer]

  for thisItem in @layers[currentLayer]
    console.log thisItem
    #console.log 'odd item' if thisItem.length != 2

    if currentLayer is 0
      console.log 'building 0 layer'
      hash = @hasher thisItem[0], thisItem[1], thisItem[2], thisItem[3]
      console.log hash
    else
      console.log 'building layer ' + currentLayer
      if thisItem[1]?
        hash = @hasher thisItem[0], thisItem[1]
      else
        hash = @hasher thisItem[0]

    #console.log hash
    pair.push hash
    console.log 'update pair'
    console.log pair.length

    if pair.length is 2
      console.log 'pushing hash'
      @layers[currentLayer + 1].push [pair[0],pair[1]]
      pair = []

  if pair.length is 1
    console.log 'pushing leftover hash'
    @layers[currentLayer + 1].push [pair[0]]
    pair = []
  console.log 'advancing layer'
  currentLayer = currentLayer + 1
  if currentLayer > 16
    throw new Error('yo')
console.log 'done'
console.log @layers

@root = @hasher @layers[@layers.length - 1][0][0], @layers[@layers.length - 1][0][1]
console.log @root
```

exports.DataLogTree = DataLogTree

Here is a truffle scenario that runs three transactions

console.log 'hello' web3Utils = require('web3-utils') VirtualToken = artifacts.require('./VirtualToken.sol') DataLogTree = require('../src/DataLogTree.js')

contract 'VirtualToken', (paccount)-> owner = paccount[0] firstPayee = paccount[1] secondPayee = paccount[2]

setUpNetwork = (options)=> return new Promise (resolve, reject)=> results = {} console.log 'creating virtual token' tree = new DataLogTree.DataLogTree() web3.eth.getBlockNumber (err, result)=> startBlock = result console.log startBlock token = await VirtualToken.new(from: owner) resolve token: token startBlock: startBlock tree: tree

it "can set crete token", -> network = await setUpNetwork() logs = await new Promise (resolve, reject)=> network.token.DataLog({},{fromBlock: network.startBlock,toBlock:'latest'}).get (err, foundLogs)=> resolve foundLogs

```
console.log logs

genesisLog = null
logs.map (o)->
  console.log o
  if o.event is 'DataLog'
    genesisLog = o
    #console.log [o.args.logType]#, web3Utils.padLeft(web3Utils.toHex(o.nonce),64), o.path, o.value]
    network.tree.addItem(o.args.path, web3Utils.padLeft(web3Utils.toHex(o.args.logType),64), web3Utils.padLeft(web3Utils.toHex(o.args.nonce),64), o.args.value)
    #console.log o.args

console.log 'building tree'
network.tree.buildTree()

console.log network.tree.root

console.log 'getting proof'
proof = network.tree.getProof(web3Utils.padLeft(web3Utils.toHex(genesisLog.args.logType),64), web3Utils.padLeft(web3Utils.toHex(genesisLog.args.nonce),64), genesisLog.args.path)
console.log proof

expectedRoot = web3Utils.soliditySha3(proof[1],proof[2],proof[3],proof[4])
```

```
assert.equal expectedRoot, network.tree.root

txn = await network.token.publishCollation(expectedRoot, "0x0")

transactionBytes = []
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("msg.sender"), web3Utils.padLeft(owner, 64))
transactionBytes.push proof
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("amount"), web3Utils.padLeft(web3Utils.toHex(1000), 64))
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("destination"), web3Utils.padLeft(firstPayee,64))
transactionBytes.push network.tree.getNullProof(web3Utils.soliditySha3(network.token.address,"balance", firstPayee))

console.log transactionBytes
myBytes = network.tree.proofBytes(transactionBytes)

console.log myBytes

console.log '["' + network.tree.proofBytesArray(myBytes).join('","') + '"]'

txn = await network.token.vTransfer.call(myBytes)

console.log ' trying second transaction ********'

logs = network.tree.parseLogs txn

console.log logs

state1Tree = new DataLogTree.DataLogTree()


for thisItem in logs
  state1Tree.addItem(thisItem[0], thisItem[1], thisItem[2], thisItem[3])

console.log 'building state1Tree'

state1Tree.buildTree()

txn = await network.token.publishCollation(state1Tree.root, "0x0")

console.log 'getting proof'
proof = state1Tree.getProof(web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.soliditySha3(network.token.address,"balance", firstPayee))
console.log proof

transactionBytes = []
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("msg.sender"), web3Utils.padLeft(firstPayee, 64))
transactionBytes.push proof
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("amount"), web3Utils.padLeft(web3Utils.toHex(500), 64))
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("destination"), web3Utils.padLeft(secondPayee,64))
transactionBytes.push network.tree.getNullProof(web3Utils.soliditySha3(network.token.address,"balance", secondPayee))

console.log 'transaction bytes for second transaction'
console.log transactionBytes

myBytes = network.tree.proofBytes(transactionBytes)


txn = await network.token.vTransfer.call(myBytes)

console.log 'second transaction results'

logs = network.tree.parseLogs txn

console.log logs

console.log ' trying third transaction **********************************************'

state2Tree = new DataLogTree.DataLogTree()

for thisItem in logs
  state2Tree.addItem(thisItem[0], thisItem[1], thisItem[2], thisItem[3])

console.log state2Tree.layers[0]

state2Tree.buildTree()

txn = await network.token.publishCollation(state2Tree.root, "0x0")

proof = state2Tree.getProof(web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.soliditySha3(network.token.address,"balance", secondPayee))
console.log proof

secondProof = state1Tree.getProof(web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.padLeft(web3Utils.toHex(1),64), web3Utils.soliditySha3(network.token.address,"balance", owner))


transactionBytes = []
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("msg.sender"), web3Utils.padLeft(secondPayee, 64))
transactionBytes.push proof
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("amount"), web3Utils.padLeft(web3Utils.toHex(250), 64))
transactionBytes.push network.tree.getVar(web3Utils.soliditySha3("destination"), web3Utils.padLeft(owner,64))
transactionBytes.push secondProof

myBytes = network.tree.proofBytes(transactionBytes)


txn = await network.token.vTransfer.call(myBytes)

logs = network.tree.parseLogs txn

console.log logs
```

assert.equal 1, 0, "hello"

Apologies for the coffeescript…it is 2.0 now so you get all the ES6 goodies.

So far only the red highlighted areas are working:

[

Scalable Stateless Contracts on Today's EVM Diagram step1

1489×1526 238 KB

](https://ethresear.ch/uploads/default/original/1X/1ee0aec6db0625876c961be27455e60550d5446a.png)

There are obviously a ton of areas that have big questions:

1. Can we create a viable consensus amongst collators?
2. How do we incentivize them?
3. How to handle no ops?
4. Have we really just put ethereum inside of ethereum and erased a lot of the checks that ethereum puts on run away gas costs?

I'm open to suggestions for 1-3.

For number four I think it is valuable to keep pushing this string. There are certainly times where one would want the current EVM to do bigger things. For example: Using this I could write an air drop contract that loads up 1,000 balances for way less gas than it would currently cost…if I can get the collators to validate the transaction for me.

*The proof format here is [length, path, type(1=add, 2=del, 3=null), nonce, value, left leaf0, right leaf0… left leafn, right leafn] where odd layers have an 0x0 on the end but the hash for those odd layers is calculated as the hash of just the left item. Open to better suggestions.