Smart Contract-enabled LightChain Test Network

Yahya Hassanzadeh-Nazarabadi*†, Kedar Kshatriya ‡, and Öznur Özkasap†
DapperLabs, Vancouver, Canada*
Department of Computer Engineering, Koç University, İstanbul, Turkey†
Department of Computer Engineering, Savitribai Phule Pune University, Pune, India ‡
yhassanzadeh13@ku.edu.tr, kshatriyakedar@gmail.com, oozkasap@ku.edu.tr

***Abstract*—*LightChain* is the first Distributed Hash Table (DHT)-based blockchain with a logarithmic asymptotic operational complexity, and a distributed storage layer, which preserves its integrity under the corrupted majority power of nodes. Running smart contract-based transactions, however, was a missing feature in the original implementation of *LightChain*. In this demo paper, we present the software architecture of our open-source smart contract-enabled test network for *LightChain*.**

***Index Terms*—Blockchain, Distributed Hash Table, Ethereum Virtual Machine, Smart Contract, Solidity.**

I. Introduction

LightChain [1] is the first Distributed Hash Table (DHT)-based blockchain with a logarithmic message complexity and storage load balancing [1]. Having a *LightChain* system with n nodes, generating and committing a single transaction or block has the message complexity of $O(\log n)$. *LightChain* also provides a DHT-based replication [2], [3], [4] mechanism for the blocks and transactions, which reduces the per node storage overhead from $O(b)$ to $O(\frac{b}{n})$, where b is the number of blocks in the blockchain. At the consensus layer, *LightChain* provides Proof-of-Validation (PoV), which is an immutable, fair, and fully decentralized consensus protocol that provides each node a uniform chance of being involved in the block generation decision-making regardless of its influence in the system, e.g., its hashing power. *LightChain* is the first blockchain architecture that preserves its integrity under the corrupted majority power of nodes [1].

With the advent of smart contracts [5], the modern blockchains can support user-defined operations at run time. This resembles invoking a function in a structured programming language. However, supporting smart contracts is a missing feature in the current local test network of *LightChain* [6], [7]. To support developing and testing smart contract-based applications, in this demo paper, we present a novel approach that enables running solidity-based smart contracts [8] on the local test network of *LightChain*. Our implementation is open-sourced at [9]. We also present a sample demo scenario of running a smart contract on the test network of *LightChain*.

II. Software Architecture

Figure 1 illustrates the life cycle of a single smart contract-based transaction through the software architecture of *LightChain* from submission to the network to being committed and effective. This life cycle happens in the nodes involving in the decision-making around the transaction and its corresponding block (i.e., PoV validators [1]). As shown by Figure 1, once a smart contract-based transaction is submitted to the *LightChain* system, it is collected by the network layer of its assigned PoV validators and is passed to their consensus layer. As part of the consensus validation, the correctness of the transaction is verified to assure that the transaction performs a valid state transition. In the original *LightChain* protocol [1] the transactions are static and self-contained, and their correctness evaluation is done by some static checks. However, in our smart contract-enabled *LightChain*, the transactions are dynamic and require execution to validate their state transition. We made the consensus layer of *LightChain* capable of executing solidity-based smart contracts on the parameters of transactions that invoke them. The execution of the transaction is done in a sandbox virtual machine environment known as the Ethereum Virtual Machine (EVM), which is used for interacting with the solidity-based smart contracts [5], [10]. Once the execution of the transaction in EVM is done, the correctness of returned result is verified by the consensus layer. If the execution passes the correctness as well as other consensus validations [1], it is marked as validated, included in a block, and passed through the storage and view layers, to get available and effective, respectively. In this *LightChain*, we present the state of a *LightChain* node by a set of state variables, represent the application-dependent asset [1] of the nodes participating in *LightChain*. State variables are solely manipulated by the smart contracts. As shown by Figure 1, once a smart contract-based transaction gets committed to the blockchain, the state variables it touches are updated accordingly at the view layer.

III. Sample Demo Scenario

Overview: In this demo scenario, we explain running simulations on *LightChain* test network by deploying solidity-based smart contracts, defining state variables for the nodes, and running test networks with smart contract-based transactions. Figure 2 shows the entire simulation pipeline in the smart contract-enabled *LightChain* test network. The entire system is encapsulated into a single docker container. Upon startup, the container executes the master thread, which reads the simulation configuration and deployed contracts, spins up the *LightChain* nodes with each node in a separate thread, executes the simulation scenario for them, and logs their interactions into an output csv file.

Simulation Configuration: The configuration file is simulation.config [9], which lists up the simulation configuration for an entire *LightChain* system. A sample configuration file

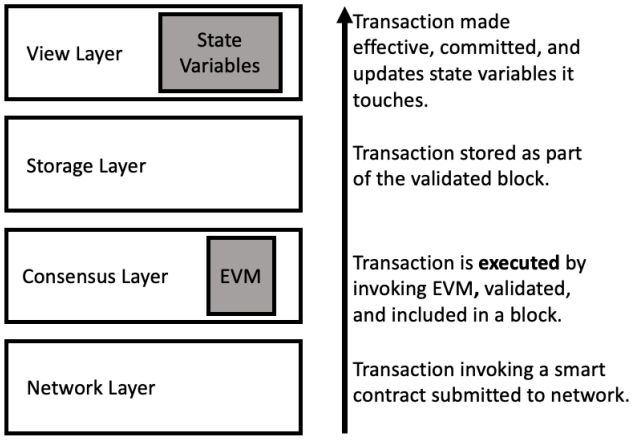


Figure 1: The life cycle of a transaction in *LightChain*, which happens on its assigned PoV validators [1]

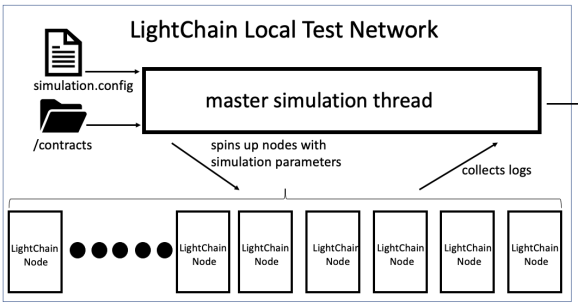


Figure 2: Simulation pipeline in *LightChain* local test network.

is shown in Listing 1, which describes a simulation scenario involving 100 nodes. Each node in this scenario generates a transaction per second for a total of 1000 transactions per node. Each block contains at least 100 transactions. Each node is initiated with 20 units of balance, which is used to pay for its transaction fees or transferring funds to the other nodes. Also, 33% of the nodes in this demo scenario are malicious nodes, i.e., they do not follow the *LightChain* protocol. The second set of parameters in this listing are the *LightChain*-specific ones, which we skip for the sake of space, and refer the interested readers to [1]. The last set of parameters in Listing 1 are the smart contract-related ones, which set the mode of simulation on the contract mode. In this demo scenario, we present the state of a *LightChain* node by a single Token state variable, which is initialized by 30 for all nodes. Additional state variables can be defined similarly. The state variables of each node are distinct and accessible using its identifier.

Smart Contracts Deployment: The deployment of smart contracts in the current version of *LightChain* test network is done manually at the compile time. The solidity-based smart contracts are placed into the contracts directory of the test network as a .sol file, and their names and functions are added to the contract.java file. More information about the deployment of smart contracts is available on our repository [9]. As our future work, we plan to make the test network capable of deploying smart contracts at the run time.

Running Test Network: The *LightChain* test network is executed by building its docker file and running the resulted docker container as specified in the repository's readme [9]. Once the test network container is started, it compiles all deployed solidity smart contracts into byte codes and starts the master thread upon the successful compilation of smart contracts. The master thread loads the simulation configurations and spins up as many nodes as the configuration file specifies with each node in a separate thread. The master thread then plays the simulation scenario for *LightChain* nodes and logs their interactions into an output csv file. Each log entry corresponds to a generated transaction or block and describes its corresponding time, memory, and bandwidth overheads. The termination of simulation happens when each node generates its specified number of transactions and all transactions are committed to the blockchain. The user can perform arbitrary queries on the logs in the output csv file, and extract the performance metrics of interest, e.g., the average transaction generation time, the average block formation time, and the average block size.

```
// simulation parameters
NODES = 100 // number of nodes in test network
TRANSACTIONS = 1000 // transactions per node
DELAY = 1 // delay between transactions of a node
BLK_SIZE = 100    // size of a block
INIT_BALANCE = 20 // initial balance of a node
MALICIOUS = 0.33  // fraction of malicious nodes

// LightChain parameters
VALID_THR = 12 // signatures threshold
SIG_THR = 10   // validation threshold
VALID_FEE = 2  // validation fee
ROUTE_FEE = 1 // routing fee
REWARD = 3    // block generation reward

// smart contract parameters
ContractMode = True // runs in contract mode
Token = 30 // initial assets of a node
```

Listing 1: Sample configuration file for running in smart contract mode.

REFERENCES

[1] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Lightchain: A dht-based blockchain for resource constrained environments," *arXiv preprint arXiv:1904.00375*, 2019.

[2] ——, "Laras: Locality aware replication algorithm for the skip graph," in *IEEE/IFIP NOMS*, 2016.

[3] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and O. Ozkasap, "Decentralized utility-and locality-aware replication for heterogeneous dht-based p2p cloud storage systems," *TPDS*, 2019.

[4] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Decentralized and locality aware replication method for dht-based p2p storage systems," in *FGCS*. Elsevier.

[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[6] Y. Hassanzadeh-Nazarabadi, N. Nayal, S. S. Hamdan, Ö. Özkasap, and A. Küpçü, "A containerized proof-of-concept implementation of lightchain system," in *ICBC*. IEEE, 2020.

[7] Y. Hassanzadeh-Nazarabadi, N. Nayal, S. S. Hamdan, A. U. Şahin, Ö. Özkasap, and A. Küpçü, "Skip graph middleware implementation," in *SRDS*. IEEE, 2020.

[8] J. Jiao, S. Kan, S.-W. Lin, D. Sanan, Y. Liu, and J. Sun, "Semantic understanding of smart contracts: executable operational semantics of solidity," in *SP*. IEEE, 2020.

[9] "Lightchain: https://github.com/yhassanzadeh13/lightchain-container."

[10] SemuxProject, "Ethereum virtual machine in java," https://github.com/semuxproject/evm, 2019.