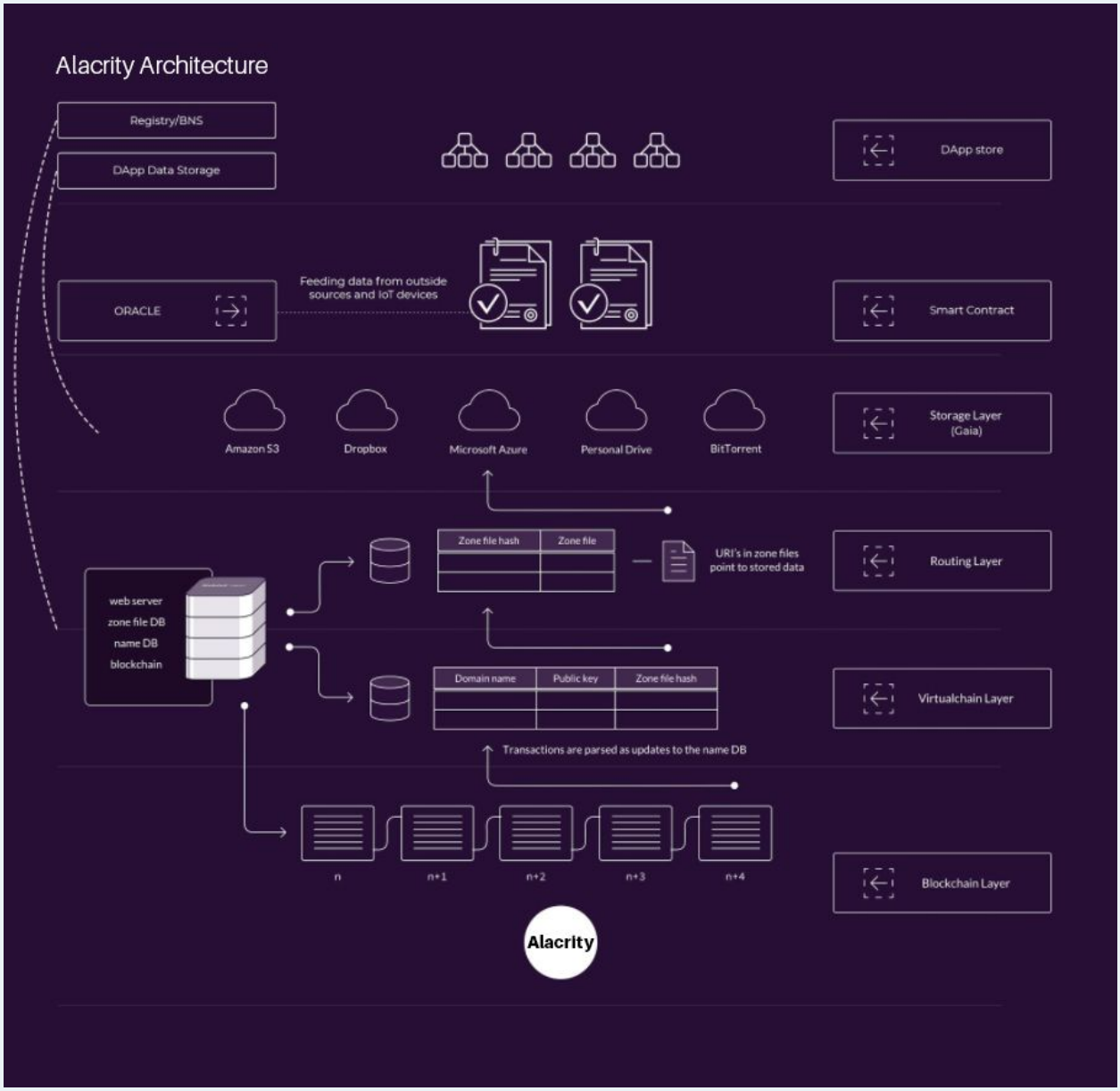
Alacrity Architecture



alacrity- a brief introduction

alacrity is a complete blockchain ecosystem. alacrity provides leverage to the Dapp Developer to establish their DApps from scratch, Using the SDK provided to them they would be able to make world-class DApps that can be used in varied use cases. The alacrity network allows them

to deploy their DApp on the alacrity blockchain which uses the graphene framework. It allows the users to make use of the decentralized storage for their Dapps which lets them enter into a whole new world of decentralization.
alacrity network would utilize its own token ALA that would facilitate the user to carry out the transaction on the blockchain network using a different token. The main takeaway of using the alacrity network is zero transaction cost which allows the users to explore the network without worrying about the monetary aspect. The Aladdin allows Dapp developers to earn their money using its unique tokenomics model.

Smart Contract

alacrity network enables the creation of smart contracts, whose execution and resource consumption is handled just like a typical application running on an operating system. The smart contract will run on the alacrity nodes, whose persistent data is stored on the node's RAM and events of actions are stored and synced on the blockchain. Smart contracts will be written in C++.

For the development of a smart contract ecosystem, alacrity network will provide

1) Local testing node
2) A method to communicate with the local node
3) Wallet and key management for accounts
4) Setup of IDE and compiler to convert smart contract in executable form, a contract development toolkit that exposes various libraries and programming constructs to aid with smart contract development. These constructs provide a programmatic interface to deal with various components of the alacrity network.

Oracle

In the blockchain space, an oracle is an entity which provides data. The need for such figures arise from the fact that blockchain applications, such as Bitcoin scripts and smart contracts cannot access and fetch directly the data they require from 3rd party sources, for example: price feeds for assets and financial application, random number generation for gambling.

alacrity network will use provable services for integration of oracles into its blockchain network as it is well known to provide oracle integration into few public blockchain like EOS, Ethereum. If any dApp is required to fetch the data from the outside world API, the dApp developer would just need to import the header file in dApp's smart contract and it will fetch all the functionality and the dApp dev can pass the API in "oraclize_query()" function.

- The "oraclize_query()" and "callback()" action handle all the communication between our alacrity contract and provable.

- The “oraclize_query()” contains special instructions which are provable to execute an off-chain task means fetching the data from a given API.
    - oraclize_query("URL","json(https://www.therocktrading.com/api/ticker/BTCEUR).result.0.last")
- According to that particular API provable will fetch the result and that transaction will execute the “callback()” and that “callback()” will be placed in the contract by its developer.
    - void callback(
        const alacrity::checksum256 queryId,
        const std::vector<uint8_t> result,
        const std::vector<uint8_t> proof
)
- Regarding our dApp store
    - alacrity platform will provide this header file as an SDK, or library to any dApp dev.
      The Dapp developer will need to implement this header file for his dApp’s smart contract so he can clone/fork in its smart contract and dev and can pass the API and fetch the data
- In smart contract how oraclize works and how oraclize_query() and how callback() will take place
    - ```
      #include "oraclize/eos_api.hpp"
      class wolframrand : public eosio::contract{
       public:
        using contract::contract;
        wolframrand(name receiver, name code, datastream<const char*> ds)
      contract(receiver, code, ds) {}
        [[eosio::action]]
        void getrandomnum(){
        oraclize_query(10, "WolframAlpha", "random number between 1 and 6");
        print(" Provable query was sent, standing by for the answer...");
        }
        [[eosio::action]]
        void callback(
          eosio::checksum256 queryId,
          std::vector<uint8_t> result,
          std::vector<uint8_t> proof
        )
        {
          require_auth(provable_cbAddress());
      ```
      ```

```
        const std::string result_str = vector_to_string(result);
        print("Result: ", result_str);
        if (result_str != "6")
          getrandomnum();
      }
};
EOSIO_DISPATCH(wolframrand, (getrandomnum)(callback))
```

Transaction Audit Trail:

Hashed transaction data will have metadata (human readable name) for audit purposes. All transaction data would be available for review at any time in the future. Audit trail will have basic search functionality to search by name.

Oracle is basically used for fetching the data from the outside world (off-chain) and sending that data to our smart contract and smart contract will process that data and then put it into on-chain.

The response of any web API ⟶ Oracle ⟶ Smart Contract

Oracle works as a bridge between the API and SC. The response of any API works as Input for Oracle and the output of the Oracle works as an Input for SC.

Blockchain

The alacrity network will be based on the graphene framework and would utilize the decentralized consensus algorithm proven qualified of meeting the performance requirements of applications on the blockchain, Delegated Proof of Stake. Under this algorithm, those who hold tokens on a blockchain adopting the alacrity network may select block producers through a continuous approval voting system. Anyone may choose to participate in block production and will be given an opportunity to produce blocks, provided they can persuade token holders to vote for them.

The alacrity network will enable blocks to be produced exactly every 0.5 second and exactly one producer is authorized to produce a block at any given point in time. If the block is not produced at the scheduled time, then the block for that time slot is skipped. When one or more blocks are skipped, there is a 0.5 or more second gap in the blockchain.

At the start of each round 15-21(still in dissent) unique block producers are chosen by the preference of votes cast by token holders. The selected producers are scheduled in an order agreed upon by 9-15(still in dissent) or more producers.

If a producer misses a block and has not produced any block within the last 24 hours they are removed from consideration until they notify the blockchain of their intention to start producing blocks again. This ensures the network operates smoothly by minimizing the number of blocks missed by not scheduling producers who are proven to be unreliable.

Under normal conditions a DPOS blockchain does not experience any forks because, rather than compete, the block producers cooperate to produce blocks. In the event there is a fork, consensus will automatically switch to the longest chain. This method works because the rate at which blocks are added to a blockchain fork is directly correlated to the percentage of block producers that share the same consensus. In other words, a blockchain fork with more producers on it will grow in length faster than one with fewer producers, because the fork with more producers will experience fewer missed blocks.

Furthermore, no block producer should be producing blocks on two forks at the same time. A block producer caught doing this will likely be voted out. Cryptographic evidence of such double-production may also be used to automatically remove abusers.

Byzantine Fault Tolerance is added to traditional DPOS by allowing all producers to sign all blocks so long as no producer signs two blocks with the same timestamp or the same block height. Once 15 producers have signed a block the block is deemed irreversible. Any byzantine producer would have to generate cryptographic evidence of their treason by signing two blocks with the same timestamp or block height. Under this model, an irreversible consensus should be reachable within 1 second.

Decentralized Storage

alacrity has gone a step ahead by introducing the use of Decentralised storage on its network. The platform would allow the users to store their data on multiple platforms in a highly secured and safe manner.

alacrity will give users control of their data using the decentralized storage system, a user-controlled storage system that allows DApp to interact with their personal data lockers. Users can host these data lockers on a cloud-provider or other data storage options like private hosting. Importantly, the user controls which service to use. Data on decentralized storage is encrypted and signed by user-controlled cryptographic keys. With the decentralized storage system, users designate the location of storage location which stores data. The blockchain will

only store the “pointers” to decentralized storage locations. When users will log in to applications and services using the protocol, they will pass that location to the application; with this information, applications know how to communicate with the specified storage locker such that the application stores data on storage specified by the user.

The decentralized storage design outlook is to reuse existing cloud providers and infrastructure in a way that end-users don’t need to trust the underlying cloud providers. Cloud storage providers (like Amazon S3) will be maintained as drives and store encrypted and/or signed data on them. The cloud providers have no visibility into the user’s data; they will only see encrypted data beads.

Further, since the associated public keys or data hashes are discoverable through the alacrity blockchain, cloud providers cannot tamper with users’ data. Writing data to a storage hub will involve pointing it to the appropriate location on that server. The hub will validate these pointing by checking that any such write request carries a signed authentication token. This token will be signed by the private key which controls the particular bucket being written to.

To accommodate separate buckets for each application a user might use, the user derives different private keys for each of those applications. Each of these private keys only grants access to specific buckets on the decentralized server. On the server, the user’s blockchain-verified routing information contains a URL that will point to a signed JSON object (signed by the owner key of the username). This signed JSON object contains URLs that point the user’s storage data locker. Once an application knows the location of the user’s data locker, they simply request a file from that location using a standard HTTP request.

Registry Layer

The alacrity registry layer will be a naming service that will bind names to off-chain state without being dependent on any centralized form of control.
The naming service will have several protocols

1) All the namespaces should be distinctive throughout the platform
2) Namespaces will have a readable form
3) Namespaces are strongly binded by the owner private keys

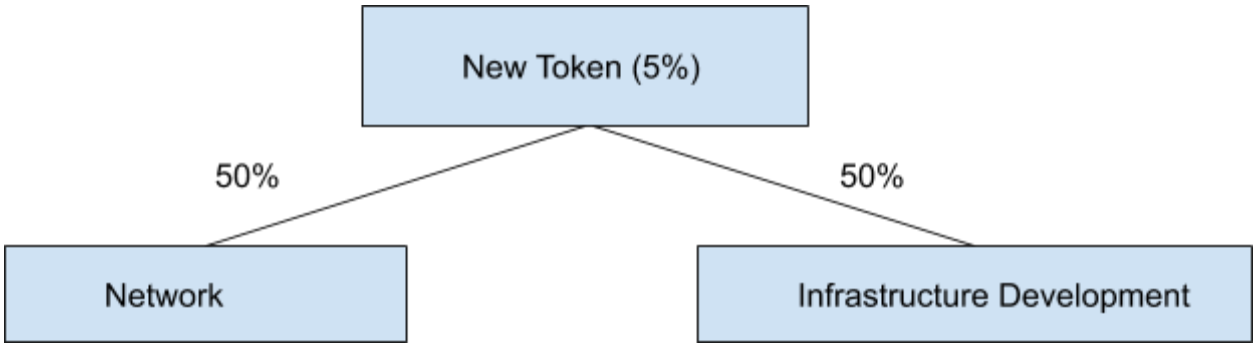
The private keys owned by the owner will generate transactions and will also update its zone file hash and its owner. The owner has to pay for the transaction on blockchain. This limits the rate of name registration. To solve this problem, the registry layer will have a subdomain whose state and owner will be stored outside the blockchain. Subdomains follow the same protocol as the naming service. This system will be maintained by a **peer-to-peer storage** system whose hashes will be on the blockchain.
The use-case of this will be follows:

1) Storing a name's routing information for its owner's datastores.
2) Storing subdomain transactions and associated state.

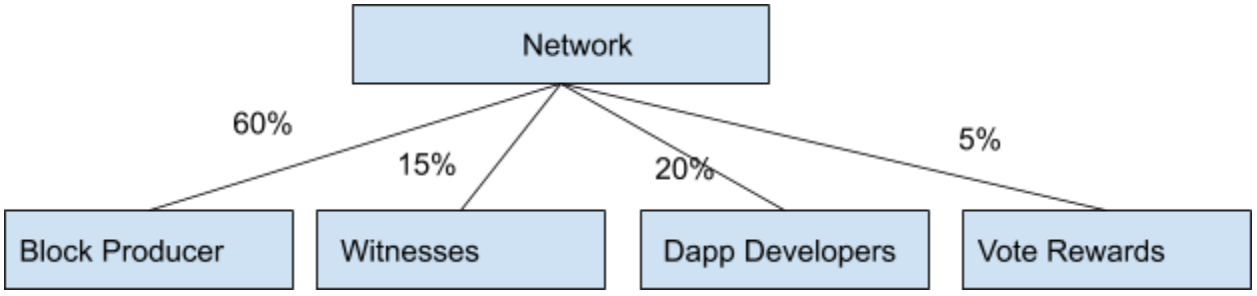
Tokenomics Model:

Token Name: **ALA**
Total Token Supply: 800 million
Inflation:5% which would decrease to 2% over a period of three years
Registration Cost: USD 20

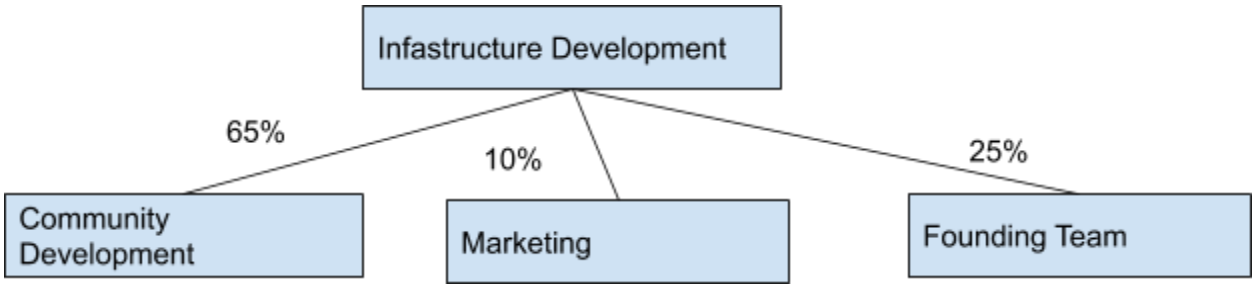
The new token would be distributed as follows:



The token distribution for the network would be as follows.



The token distribution for the Infrastructure Development would be as follows:



Block Producer Reward

The 21 Block Producer on the network would get 60% of the total token allotted for the network. The 60% would be equally divided amongst the block producer, this particular pay-out would occur daily

Witness Rewards

The Top 21 block and the remaining 29 back-up block producers would get 15% of the total token supply alloted to the network part. This pay-out would occur on a daily basis.

Vote Reward

The vote reward will be divided amongst the users who would vote in the Aladdin Community.

Dapp Developer

The Dapp Developer would be divided into two portions.

Number of Users

The Dapp developer would get rewards based on the number of unique users they onboard onto their Dapp.

Incoming Volume

The Dapp Developers would get rewards based on the volume of incoming token happening on their Dapp