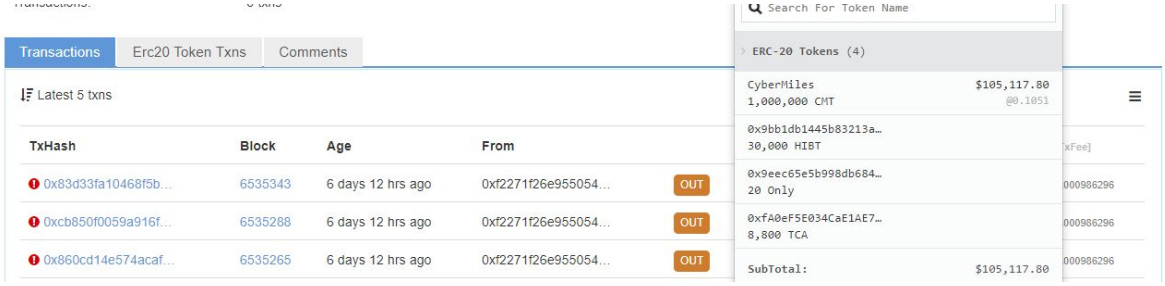
Logos Network Token Whitepaper

Draft Carl Hua 10/20/2018

The intention of this white paper is to provide a summary of Logos Network's token capability. This white paper serves as the first level technical input for requirement derivation. Before we dive into the Logos' token design, here are some backgrounds on token support from other platforms:

Ethereum:

Contrary to popular belief, there is no native token support on Ethereum platform. All token support is realized via smart contracts. ERC20 token standard is a merely a guideline to program a smart contract fo follow certain rules. As of today, there are overwhelmingly large amount of custom ERC20 token implementations from all over the web that claim to stay true to the original vision - however we are seeing unorthodox implementations coming out of even the largest players in the market. ERC20 guidelines do not provide a capability for the custodian or token issuer to "take back" tokens, in the most recent deployment of Gemini ERC20 contract, they included code to "take back" tokens as they wish - which went unnoticed for a while. This is due to compliance policy and regulation needs with the regulators, but the takeaway here is that there is no single true ERC20 token implementation in this industry. All tokens live within the smart contract's storage - in a huge array type of data structure that contains a mapping between account public key and token amount. When a send request is sent to the smart contract, the storage data is updated and the token balance is also updated. In other words, although Ethereum employs an account based approach, there is no native understanding of token concept by the first layer solution. Ethereum relies on the blockchain explorer to interpret all of smart contract's data to provide an illusion that accounts own tokens. The diagram below shows that the blockchain explorer displays an account's token balance - however in order to do so, the blockchain explorer must parse the entire ethereum database. As of today, it is at around 670GB.



Ethereum's storage uses Merkle Patricia trie to facilitate access to data at the cost of complexity to save storage space. This design organically permutates to the ERC20 smart contracts as well - and in other words if a smart contract has 50 million accounts, it would take up 50 million accounts worth of space inside the smart contract's storage in addition to the 50 million accounts record in the first layer. As you can see, this quickly becomes a scalability issue.

Stellar:

Stellar

Stellar does not have turing complete smart contract capability. Each and every operation is bounded and implemented in native C++. The support of token is realized via the issuance of assets. In lieu of a complete storage solution similar to Ethereum's Merkle Patricia trie with key/value store backend, Stellar uses SQLite relational database as its backend. For those of you who are interested in the performance degradation, there is detailed benchmark comparison here http://www.lmdb.tech/bench/microbench/benchmark.html

To summarize Stellar's asset capability, we can think of it as a pre-designed and programmed ERC20 contract. The upside for such design is performance at the execution layer (minus the database backend part) while sacrificing flexibility in the design of the token protocol and the upgradability without pushing out a new version of the core software.

With the bounded operations you can perform on the account (see their developer's guide for more details https://www.stellar.org/developers/guides/issuing-assets.html), developers can create second layer solutions on top of the bounded operations to achieve additional goals that aren't supported by the first layer solution. The philosophical difference between Ethereum and Stellar is that the second layer solution is uploaded to the blockchain for Ethereum while Stellar embraces a trust model that allows the token issuance to have greater performance.

OmniLayer

OmniLayer is a second layer solution that lives on top of bitcoin to realize token issuance. It piggybacks Bitcoin's protocol and use data within the messages to build an additional layer of capacity. OmniLayer is slow and not suitable for Logos Network.

NEO

Similar to Ethereum, NEP-5 Tokens are smart contracts that live on the blockchain. It suffers the same type of issues with ERC-20 and provide zero innovation. We can generally consider NEO is a copy cat of Ethereum with extremely centralized governance.

Logos Network

Logos Network provides native token support without the need for smart contract. The proposed solution is a hybrid approach to better suit Logos Network's existing architecture.

Similar to Ethereum, there will be a "token account" (or another name) for each of the tokens that is issued on Logos Network. Within a token account, it contains vital information regarding to the token, this information includes token label, total supply, token settings, issuer's information and other information that is needed to facilitate the execution of token transfer.

Some of the settings are immutable while others can be updated. Token accounts have their own account chain similar to regular account's account chains in the sense that each update to the token account is recorded and logged in the network. All of token account's updates are run through delegate's consensus. The details of the settings are omitted in this paper and will be added at a later time. In summary, Logos Network supports both fungible tokens and non-fungible tokens. The token issuer can choose to charge a token transfer fee that is payable to a selected account in addition to the conventional transaction fee charged in Logos currency.

Note that most of the settings are all contained within the token account with the exception of whitelisting. If whitelisting capability is turned on for the token, the receiver of the token must submit a whitelist request via delegate consensus, and it is when such request is processed, the token account controller can whitelist the account by toggling the "isAbletoRecieve" flag to true.

As part of the token support effort, Logos Network account's capability are expanded to understand token concepts. When an account receives a token, the account is updated to include such token balance in the database. Additional operations are created at the core protocol level to understand the transfer of tokens. The operations behave similar to send operations in that the actions are recorded on the account chain and is permanently saved in the network.

This token approach is similar to Ethereum's ERC20 token in the sense that we have a dedicated token account that provides necessary information for the assest. However, Logos Network does not use smart contract to actually execute the transfer logic and other custodian activities. For transfer logic and other custodian activities, Logos Network is similar to Stellar that the actual execution implementation is realized via core protocol.

The backend implementation uses LMDB to realize the design and this remains an area of active research as scalability is still a concern. The initial reaction was to have the whitelist as part of the token account, this will quickly become an issue if the whitelist becomes large - key/value store does not play well with large sets of data. Although we can store key value pointers that points to a list of keys to reduce the token account size for whitelist, it is not an ideal solution. This is the reason why we require account holders to submit a token whitelist request so that the token issuer can whitelist accounts without needed additional storage. We rely on the blockchain explorer to parse those data for the final list of whitelists.

There are many settings needed in the token account that will be detailed in the requirements. The intention of this paper is to give an overview of the design of the token platform. Here are some use cases that will attempt to give an overview of how the token platform works at the implementation level:

A token issuer decides to issue a fungible stablecoin called Carl's MasterCoin, in this case we will call it CMC. Carl would initiate a token issuance request to the delegates with the proper settings: token name, total supply, token controller, blacklist addresses, token issuer's name,

address and email, fungible token, revocable and whitelist required. Carl pays a respectable amount of fee in Logos and upon the successful consensus execution, a token account is created with the hash address returned to Carl. At this point, LMDB is updated with a token account that contains those information in binary.

Carl issues a command to update the CMC's controller account pubkey to 0xABC - again the controller account is now updated with the new account pubkey and now only the new controller account pubkey can operate on the token account.

Devon would like to receive CMC, so Devon submits a request into the network to be whitelisted. Devon provides the hash of the token account and upon the successful execution of the request, Devon's account now has CMC with a balance of 0. Whitelist flag is false since Carl has yet to approve the whitelist. At this stage, LMDB is updated to create an entry for CMC in Devon's account.

Carl whitelists Devon's account by sending a request to the network - it gets rejected because Carl is no longer the controller of the token account. Transaction fee is only taken when a request is processed.

Carl realizes his mistake, and contacts the controller 0xABC, 0xABC whitelists Devon's account. An operation is performed on the token account - with a new request pended to the account chain. Devon's account's whitelist field of CMC is toggled on, now Devon is able to receive tokens. Blockchain explorer receives these operations, adds Devon's account to CMC's whitelist.

0xABC sends some CMC to Devon's account - balances are deducted from CMC's token account and added to Devon's account.

Devon then commits a crime and FBI contacts 0xABC to revoke Devon's balance.

0xABC has two options:

1. Send a revoke command to the network - Devon's balance is returned to CMC's token account
2. Send a blacklist(freezelist) command to the network - Devon can no longer send such funds. This pause flag lives in Devon's account. Note that the token account will have Devon's address added and no other transaction can interact with Devon's account. This is done in such a way so that the token account can blacklist accounts without the account having a balance to begin with.

Devon then was proven innocent - it's time to release his funds.

0xABC would send a whitelist command to the network - Devon can send/receive funds again.

0xABC can also send a command to the network to create CMC since it is a fungible token - the total supply would increase and a request is appended to the token account.

As we can see, most operations related to the token account is appended to the token account chain while most operations related to send/receive of such token is appended to the token account holder's account chain.

Database layout remains to be a concern and should be thoroughly vetted (after mainnet launch?).