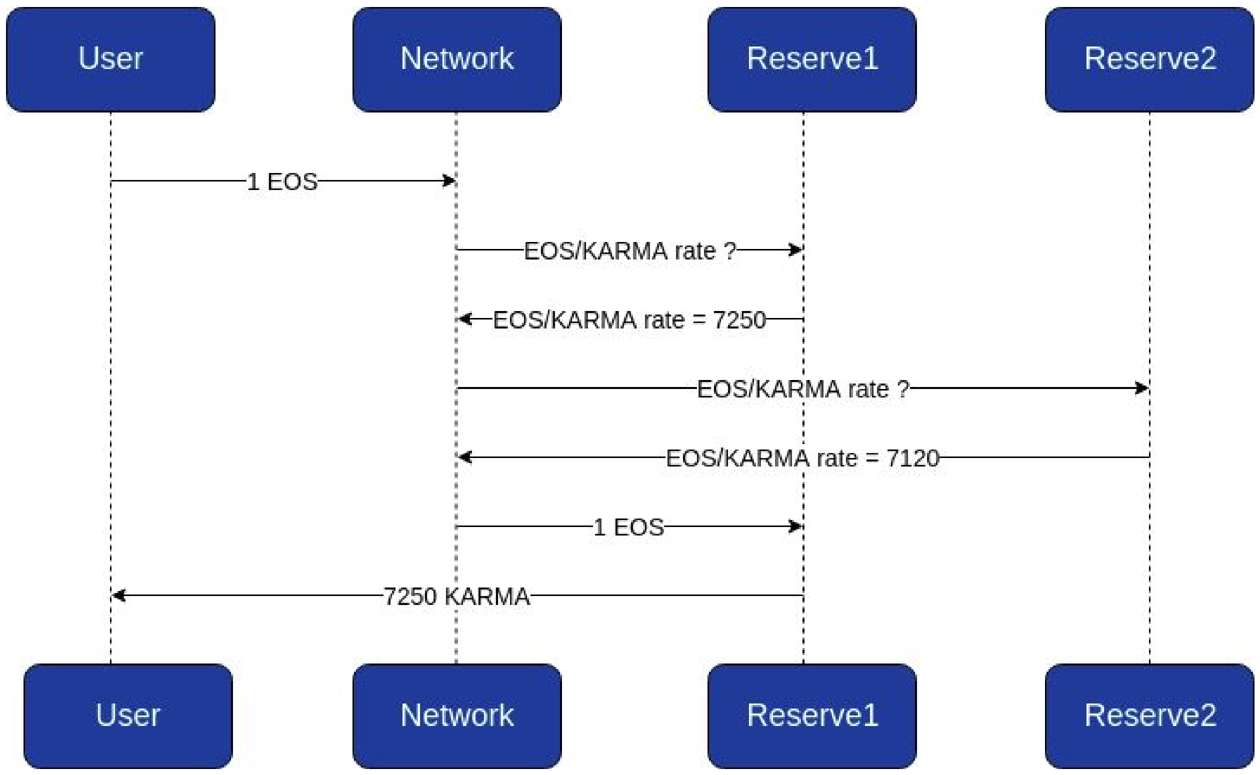
YOLO Technical Specification

Introduction

This document describes the current implementation and architecture of the YOLO decentralized exchange (DEX).

YOLO smart contracts allow users to send one type of token (e.g. EOS) and receive a different token in return (e.g. KARMA) according to the conversion rates set by the reserve(s). The entire process happens in a single transaction so at no point does the contract hold users' funds. A conversion between EOS and KARMA is depicted in the diagram below:



The conversion is possible thanks to reserve contracts that hold inventories of tokens and EOS and provide conversion rates to the network contract. The inventory of every reserve, is managed by a reserve admin.

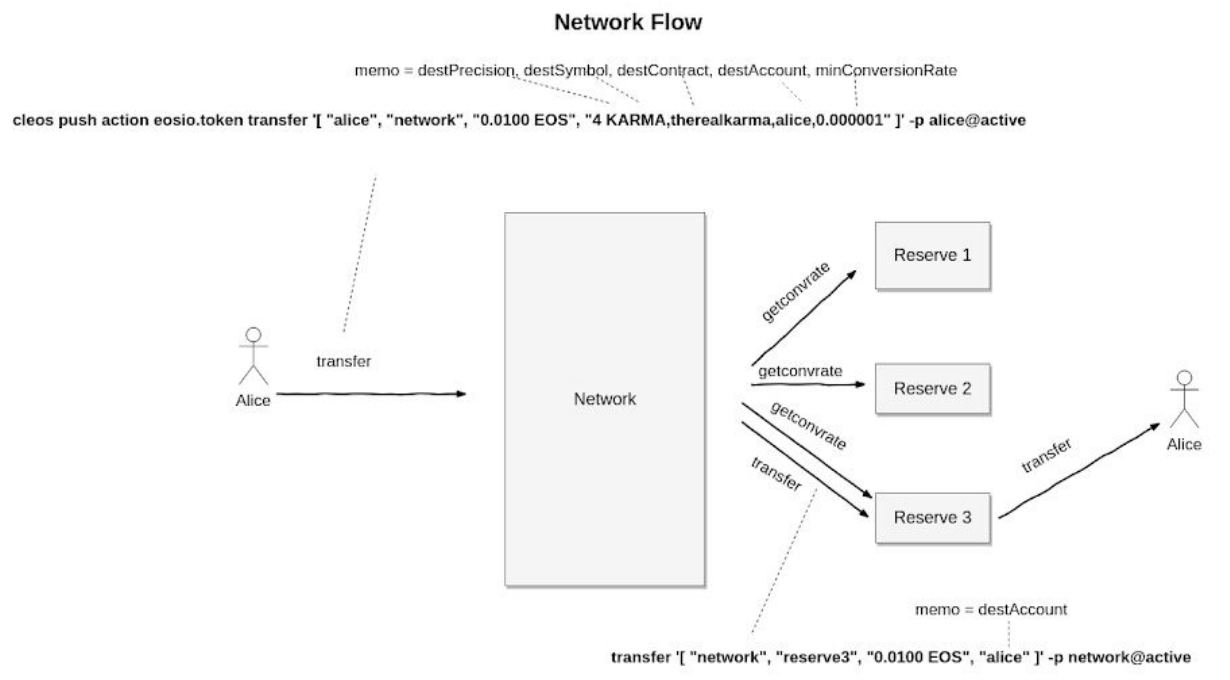
The users do not need to pay any additional fees, apart from cpu and bandwidth of a regular EOS tx. Platform fees might be paid by the reserve that executes the exchange, however YOLO will launch without such fees.
There is also no user data stored at the network or reserve contract, so no need for expensive per user RAM allocation.

Upon launch the DEX will support trades of mainnet EOS to mainnet based tokens, and vice-versa. Support for token to token trades will be added in a later stage.

This is the current YOLO testnet deployment on Jungle2:
https://dev-yolo.knstats.com/

Detailed Exchange Flow

When a trade is executed, the Network.cpp contract queries rates from all the reserves offering the required token (i.e., calls AmmReserve.cpp). From the conversion rate, the token amount of the exchange is estimated.
The Network.cpp contract transfers user's funds to the reserve with the best rate, and eventually verifies the the reserve indeed transferred back to the user.



```
cleos push action eosio.token transfer '[ "alice", "network", "0.0100 EOS", "4 KARMA,therealkarma,alice,0.000001" ]' -p alice@active
```

```
transfer '[ "network", "reserve3", "0.0100 EOS", "alice" ]' -p network@active
```

Reserves

The reserve’s role is to execute exchanges and provide rates for Network.cpp. The reserve contract has no direct interaction with the end users apart from sending funds to destination.
Each reserve is deployed by an external project to YOLO, usually token projects or market makers hired on token projects behalf. The deployment usually uses reserve code supplied by YOLO.
The reserve funds and owner/active account/keys are in the hands of the external project. However YOLO might help with deployment, account management and reserve funds.

AMM Reserve

Many reserve models might exist, such as order book based reserves or reserves based on live price feeds. YOLO will initially launch with one single model - Automated Market Making (AMM) Reserve.

The automated reserve was created with ease of maintenance as a top consideration. For the automated price reserve model, the reserve manager relies on the smart contract and its predefined algorithm to automatically provide rates for the token being served liquidity for. In this model, there are no server or development costs. This automated model only supports one token for one reserve. If one wants to list another token using this model, another automated reserve must be deployed.

Moreover, the automated reserve was also designed to help discover the price of a newly created token. Through the interaction of buyers and sellers, an estimated market price is discovered based on the market's sentiment at a point in time.

In essence, and similar to the Bancor model (but with different math theory) the automated reserve manages prices based on inventory of EOS. When a user buys the token, the reserve gets more EOS in return. In such a scenario the token price will go higher. Similarly, when user trades the token back to the reserve the EOS inventory goes lower and the token price goes lower as well.

When either EOS or token is depleted, the reserve admin can replenish the inventory and set new parameters to allow the token trading to continue.

Liquidity Pricing Formulas

The AMM reserve rates are based on the liquidity.hpp module/logic, which uses the following factors for the calculation:

- E - The contract's amount of EOS in the time of trade.
- Configured liquidity parameters:
  - Pmin - minimum price allowed.
  - r - liquidity Rate, the amount of price change per 1 EOS worth of reserve inventory change. This parameter is correlated to the initial EOS amount in the reserve and Pmin. Thus it should be configured wisely.
- The trade src amount - denoted as ΔE for a buy trade and ΔT for a sale trade.

Using the above notations, the price formulas are:

1. **Sell price for 0 src quantity:**

   $$\boxed{P(E) = P_{min} e^{r \cdot E}}$$

2. **Buy price for 0 src quantity:**

   1 / P(E)

3. **Sell price for non 0 src quantity:**
   Amount of tokens to receive for ΔE src EOS:

$$\Delta T(\Delta E) = \frac{1}{r \cdot P(E)} \cdot \left(e^{-r\Delta E} - 1\right)$$

Actual sell price is ΔE / ΔT

4. **Buy price for non 0 src quantity:**

   Amount of EOS to receive for ΔT src tokens:

   $$\Delta E(\Delta T) = -\frac{1}{r} \ln(1 + r \cdot P(E) \cdot \Delta T)$$

   Actual buy price is ΔT / ΔE

Note that the prices for 0 src quantity are mostly used for displaying a default rate in the exchange website, but are usually not used in practice.

Profit and Fees

The AMM reserve supports configuring a profit percent, which will be taken into account when calculating the rate. The profit is always calculated as EOS tokens is sent to a reserve configured fee wallet. This profit is the property of the reserve manager (token project).

There is also an optional RAM fee. This is a fixed eos amount which the reserve manager can configure to be saved on each trade. Its purpose is to potentially cover for the cost of transferring tokens to accounts which do not hold them yet (and therefore require an allocation of RAM line for balance).

While YOLO will launch without system fees, there might be such fees added in the future. Such fees might be paid by reserves for using the system, according to their volume. If/when such a mechanism is introduced, it is advised for the reserves to use the profit percent parameter to accumulate the relevant amount needed for paying fees to the system.

Listener

The listener contract is a separate contract designed for handling network hooks, such as rebate promotions, fee burning (if/when such mechanism is introduced), additional stats calculations. We initially use it for a rebate promotion.

Account Management Roles

Both the network contract and reserve contract have two roles:

1. Contract account owner (part of EOS permission system, controls the accounts owner/active authorities).
2. Contract admin (configured in the contract).

The account owner is responsible for contract deployment and possibly initial funding.
The admin is responsible for parameter configuration and ongoing funding (if needed).
The above separation supports a situation where even if the account ownership is forfeited (temporarily or permanently) the contract is still operational by the admin. Such forfeiting might be needed because of EOS support for changing contracts code (see Trust and Security Model below).

Trust and Security Model

Who should the user trust?

As the YOLO protocol does not hold the users' funds, there is no need for the user to trust us. Additionally, since our protocol is also entirely on-chain, users are able to freely verify and audit our smart contracts (either by comparing hash of github code to the deployed code hash or possibly by using on-line verification and publishing methods such as in Ethereum's etherscan).

Having stated the above, the EOS system does allow contracts to be seamlessly redeployed to the same account name. Therefore there could theoretically be a situation where the network code can change without user knowing (or right after he sent a trade tx and before that tx was accepted).

A possible solution for the above trust model problem might be revoking control of the network contract, either permanently, or for a certain and expandable amount of time (see https://github.com/xJonathanLEI/eosyield).

As such solutions are not yet common practice in the EOS community, we will first launch while having control of the network contract, and consider changing to one of the above solutions in a later change.

Who should YOLO Network trust?

For the YOLO protocol to be fully functional, there needs to be at least one reserve offering liquidity at all times.

Note that the network contract has safeguards protecting the trade funds from malicious reserves:

- The min conversion rate parameter can not be bigger than the conversion rate returned from the chosen reserve.
- The returned reserve rate is translated to an expected destination quantity/asset. The network reverts if such an amount is not added to the destination account (receiver) by the end of the trade.

Who should the reserves trust?

The reserves are required to trust the YOLO Network admin. While reserves' funds are not at risk, YOLO admin has the ability to halt a reserve's operations within the platform. In addition, the reserve managers should be aware that their reserves could be affected by extreme market conditions like flash crashes or from sub-optimal inventory management (e.g. setting wrong prices or from large exposure to risky tokens).