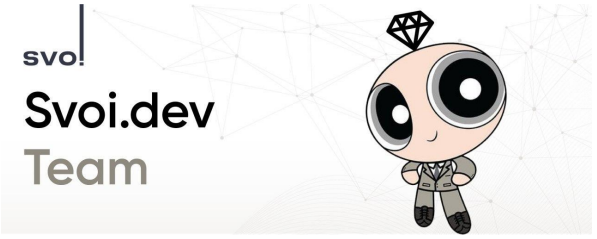


Russian version below.

CONTENTS:

PRESENTATION

Hello everyone!
We are SVOI.dev team and in this project we present the results of our long patient work,
submission to the contest:
#23 FreeTon DEX Implementation Stage 2.
The purpose of this work was the development and implementation of a fast exchange (Swap) in the FreeTON blockchain and infrastructure based on the “liquidity pool” approach.

Link to the project website: https://swap.tonswap.com
Link to the LP Explorer: https://explorer.tonswap.com
GitHub:
https://github.com/SVOIcom/tonswap-SC
https://github.com/SVOIcom/ton-testing-suite
https://github.com/SVOIcom/tonswap-frontend
https://github.com/SVOIcom/tonswap-explorer
https://github.com/SVOIcom/debots-collection

Wallet: 0:0439186aa0147661ebaf2b32ecc76bac172fcdaa24c7df7c9cb03cc816e435e6

Contact information

Telegram (corporate account): https://t.me/svoidev
team’s website: https://SVOI.dev

Basic economic model and description of cash flow in the TonSwap system

As a result of the implementation of the TonSwap project presented by our team, an architecture for the implementation of a fast exchange (Swap) was created in the FreeTON blockchain and an infrastructure based on the general concept:

Automated Market Maker (AMM) using constant product formula.

Automated market makers are the backbone of the DeFi space. They enable just about anyone to create markets easily and efficiently. While they have their limitations when compared to order exchanges, the overall innovation they bring to cryptocurrency is invaluable.

AMM exchanges overcome the concept of an order book. Instead of setting prices by demand and supply, an AMM pools liquidity together and sets prices by way of a deterministic pricing formula.

We believe that this particular solution in terms of the DEX architecture is the most reasonable, profitable and effective for use in the TON network.

Let's see from the economic perspective how such a model can incentivize users to use it for their daily exchange operations and at the same time benefit on the growth of the network in general and exchange in particular.

The basic motivation is obvious, people want to transfer their assets in blockchain depending on their needs and purposes, they want to exchange assets and in the best case scenario to earn yield rewards for providing liquidity.

On the other hand a good DEX implementation should ensure the stability of the exchange process, gain some dividends for active users and liquidity providers and provide some governance mechanism for future changes and improvements.

AMM (for ordinary users) can be thought of as a robot that is always ready to offer a price between two assets.

With the help of AMM, you can safely trade, as well as become the host of pairs / pools yourself, which allows almost anyone to become a market maker on the exchange and receive a commission for providing liquidity.

In simple terms, the TonSwap functioning model looks the next way:

1. liquidity providers place a pair of tokens - in existing liquidity pools or creating new ones;

- if a new pair is created, the primary liquidity provider sets the exchange rate of one token from the pair to another;
- in the case when the liquidity provider contributes tokens to an already existing pair (formed pool of liquidity) - tokens must be deposited into the pool in accordance with the exchange rate between the tokens of the pair that exists at the time the tokens are deposited into the pool;

2. when suppliers provide liquidity (a pair of tokens), they receive "token" of the pair (Defined as the ratio `user-contributed liquidity` /` total liquidity`) - in confirmation of the share of participation in the liquidity pool;
3. traders / buyers, through the created Swap, can exchange their tokens for other tokens within the existing pairs at the rate calculated based on token amount in the swap pair/liquidity pool at the time of the operation;
4. When performing an exchange, a trader pays a commission, which is distributed between the liquidity providers of the pool in which the exchange took place.
   The commission is distributed in accordance with the shares of the liquidity providers.
   The awards are determined by protocol. Currently, the commission is 0.3%, following the example of Uniswap v2, which may later change. Other platforms or forks may charge lower fees in order to attract more liquidity providers to their pool.
5. when changing one token to another in the established pool, their ratio within the pool changes depending on the volume of entered and contributed tokens;

* the implementation of an exchange transaction is possible in the case when the buyer of tokens in the wallet has a sufficient volume of another type of tokens of the pair for exchange, as well as to cover the commission paid to liquidity providers;

6. The best condition for swap pair functioning is when swap operations do not have a large impact on token price. For this certain conditions are required:

- there are a large number of liquidity providers and buyers / traders
- the exchange operations are carried out in small amounts compared to the liquidity pool volume.

But these are conditions that cannot be directly influenced.

7. It is important to describe one more of the actions carried out within the Swap Pair - withdrawal of the provided tokens by the liquidity provider:

if at a certain moment the liquidity provider decides to withdraw his tokens from the pool, then at the moment he owns an amount of tokens not equal to the initial one, but equal to its share of the total volume.

For example:

The liquidity provider N initially invests in the pool a pair of tokens - 1 WETH and 1 WBTC - based on the established rate of 1: 1.

At the time of entering the pool, 9 WETH and 9 WBTC have already been invested, thus his share in the pool is 10%.

Suppose, for some time, there are a lot of exchange operations performed , as a result of which we get the following ratio of the pair's tokens:

6 WETH and 20 WBTC.

If, during the time period of this established ratio, the liquidity provider N decides to withdraw his tokens from the pool, then he receives them in the amount:

0.6 WETH and 2 WBTC.

The Efficiency of Automated Market Makers

In AMM models, the price is determined by deterministic formulas based on the supply and demand of the assets. As a result, these models rely on the arbitrageurs to level the price with the other markets. In other words, when there is a price difference, the arbitrageurs start buying/selling assets from the smart contract, which changes the supply and demand of an asset and levels the price. However, this results in inefficiency for liquidity providers as they endure losses (or opportunity costs relative to the true market price that the arbitrageurs utilize).

All of the above can be presented in the form of a basic formula:

We have a market which is for trading coins of type α for coins of type β (and vice versa). This market has reserves Rα > 0 and Rβ > 0, constant product k = RαRβ, and percentage fee (1 − γ). A transaction in this market, trading ∆β > 0 coins β for ∆α > 0 coins α, must satisfy

$$(R_\alpha - \Delta_\alpha)(R_\beta + \gamma\Delta_\beta) = k.$$

After each transaction, the reserves are updated in the following way: Rα "→ Rα − ∆α, Rβ "→ Rβ + ∆β, and k "→ (Rα − ∆α)(Rβ + ∆β). We will always require that Rα,Rβ > 0, such that any trade that results in a nonpositive reserve is never fulfilled (equivalently, we say that such a trade has infinite cost).

When the fee is zero (i.e., γ = 1), any trade ∆β to ∆α must change the reserves in such a way that the product RαRβ remains equal to the constant k.

Detailed technical description of the proposed implementation with the rationale for the chosen approach: smart contracts, level of integration, interfaces

Developed Smart Contracts:

1. SwapPairRootContract;
2. SwapPairContract;
3. TIP3TokenDeployer.

Auxiliary Contracts Used:

1. TONTokenWallet;
2. RootTokenContract.

Description of developed Smart Contracts

SwapPairRootContract

Description of the smart contract

This smart contract is used to account for the created swap pairs, as well as to create new unique swap pairs.

Description of smart contract functions

The functions of this smart contract that are of any interest to developers are described in the next section. Undescribed functions are service functions and are not intended for calls from outside (modifiers or internal functions).

Description of smart contract interfaces

IRootSwapPairContract interface

- deploySwapPair (address tokenRootContract1, address tokenRootContract2) external returns (address)
  Function parameters:

1. tokenRootContract1 - address of the first root contract of the token;
2. tokenRootContract2 - address of the second root contract of the token;
   Returned values: address of the future swap pair;
   Function description: This function is used to create new swap pairs. To create a new swap pair, it is required, together with the function call, to send the required number of TONs that will be used to initialize the swap pair. The exact value of TONs depends on the root contract.

- *checkIfPairExists (address tokenRootContract1, address tokenRootContract2) external view returns (bool)*
  Function parameters:

1. tokenRootContract1 - address of the first root contract of the token;
2. tokenRootContract2 - address of the second root contract of the token;
   Returned values: is there a swap pair for these tokens or not;

   Function description: it is checked whether a swap pair exists for a given pair of tokens or not;

- *getServiceInformation () external view returns (ServiceInfo)*
  Returned values: service information about the root contract;
  Description of the function: receiving service information about the root contract;
- *getAllSwapPairsID () external view returns (uint256 [] ids):*
  Returned values: an array of all existing swap-pair IDs;
  Description of the function: this function returns the unique IDs of all swap pairs created through this root contract.

- *getPairInfoByID (uint256 uniqueID) external view returns (SwapPairInfo swapPairInfo)*

Function parameters:

1. uniqueID - unique identifier for the function;

Returned values: information about the swap pair;

Description of the function: this function returns information about a swap pair by its unique ID. Since the process of initializing the swap pair takes some time, when requesting information immediately after creating a swap pair, the information received will be incomplete. It is necessary to wait for the completion of the initialization of the swap pair.

- *getPairInfo (address tokenRootContract1, address tokenRootContract2) external view returns (SwapPairInfo)*

Function parameters:

1. tokenRootContract1 - address of the first root contract of the token;
2. tokenRootContract2 - address of the second root contract of the token;

Returned values: information about the swap pair;

Function description: similar to the getPairInfoByID function, however, instead of using the unique ID of the swap pair, the addresses of the TIP-3 token root contracts for which the swap pair was created are used.

- *setTIP3DeployerAddress (address tip3Deployer_) external*

Function parameters:

1. tip3Deployer_ - address of the TIP-3 root contracts deployer;

Function description: this function sets the address of the smart contract used to deploy new TIP-3 tokens. This contract is used during the initialization of the swap pair to create LP tokens.

- *getFutureSwapPairAddress (address tokenRootContract1, address tokenRootContract2) external view returns (address)*

Function parameters:

1. tokenRootContract1 - address of the first root contract of the token;
2. tokenRootContract2 - address of the second root contract of the token;

Returned values: future address of the swap pair;

Description of the function: this function is used to obtain the future address of the swap pair based on the transmitted addresses of the root contracts of TIP-3 tokens.

IRootSwapPairUpgradablePairCode interface

- setSwapPairCode (TvmCell code, uint32 codeVersion) external

Function parameters:

1. code - new swap pair code;
2. codeVersion - version of the smart contract code;

Function description: this function is used to set a new smart contract code that will be used to create new swap pairs. Also, the new code can be used to upgrade existing swap pairs.

- *upgradeSwapPair (address tokenRootContract1, address tokenRootContract2) external*

Function parameters:

1. tokenRootContract1 - address of the first root contract of the token;
2. tokenRootContract2 - address of the second root contract of the token;

Description of the function: this function is used to upgrade existing swap pairs. To start the update, you need to attach 3 TONs to the sent message.

IRootContractCallback interface

- *swapPairInitializedCallback (SwapPairInfo spi) external*

Function parameters:

1. spi - information about the swap pair received from the swap pair itself after the initialization process.

Description of the function: this function is used by swap pairs in order to transfer updated information containing the addresses of the created wallets and LP-token. Can only be triggered by a swap pair created through this root contract.

Description of events created by the swap-pair root contract

- DeploySwapPair (address swapPairAddress, address tokenRootContract1, address tokenRootContract2) - this event is generated when a new swap pair is created;
- SwapPairInitialized (address swapPairAddress) - this event is generated when the swap pair completes the initialization stage and is ready to function.

TIP3TokenDeployer

Description of the smart contract

This smart contract implements the liquidity pair mechanism.

Description of the functions of a smart contract

The functions of this smart contract that are of any interest to developers are described in the next section. Undescribed functions are service functions and are not intended for calls from outside (modifiers or internal functions). In this case, the smart contract does not have direct access to the functions of the swap pair, such as the functions of exchange, deposit and withdrawal of liquidity. These functions are called by transferring to wallets belonging to a smart contract of a swap pair with a special payload created in advance. This payload must be formed by calling functions belonging to the swap pair (they are described below).

Description of the interfaces of the smart contract

ISwapPairContract interface

- *getExchangeRate (address swappableTokenRoot, uint128 swappableTokenAmount) external responsible view returns (SwapInfo _swapInfo)*

Function parameters:

1. swappableTokenRoot - the address of the smart contract, the tokens of which will be provided for the exchange operation;
2. swappableTokenAmount - the number of tokens provided for the token exchange operation;

Returned value: information about the token exchange operation

Description of the function: this function is used to obtain information about the result of the exchange, if the exchange occurs at the current moment;

- *getCurrentExchangeRate() external responsible view returns (LiquidityPoolsInfo lpi)*

Returned value: information about the current state of the swap-pair liquidity pools;
Description of the function: this function returns the current state of the swap-pair liquidity pools;

- *getPairInfo () external responsible view returns (SwapPairInfo info)*

Returned value: information about the swap pair;
Description of the function: this function returns information about the swap pair;

- *getWithdrawingLiquidityInfo (uint256 liquidityTokensAmount) external view returns (uint128 withdrawedFirstTokenAmount, uint128 withdrawedSecondTokenAmount)*

Function parameters:

1. liquidityTokensAmount - the withdrawn amount of LP tokens;

Return values:

1. withdrawedFirstTokenAmount - how many TIP-3 tokens of the first TIP-3 root contract will be withdrawn;
2. withdrawedSecondTokenAmount - how many TIP-3 tokens of the second TIP-3 root contract will be withdrawn;

Description of the function: how many TIP-3 tokens will be received when withdrawing liquidityTokensAmount LP-tokens from the swap pair at the moment;

- *getProvidingLiquidityInfo (uint128 maxFirstTokenAmount, uint128 maxSecondTokenAmount) external view returns (uint128 providedFirstTokenAmount, uint128 providedSecondTokenAmount)*

Function parameters:

1. maxFirstTokenAmount - how many tokens of the first TIP-3 contract will be deposited;
2. maxSecondTokenAmount - how many tokens of the second TIP-3 contract will be deposited;

Return values:

1. providedFirstTokenAmount - how many first TIP-3 tokens will be entered if the withdrawal occurs at the moment;
2. providedSecondTokenAmount - how many second TIP-3 tokens will be entered if the withdrawal occurs at the moment;

Description of the function: this function allows you to determine how many TIP-3 tokens will be deposited if the liquidity is entered at the moment.

- *getAmotherTokenProvidingAmount (address providingTokenRoot, uint128 providingTokenAmount) external view returns (uint128 anotherTokenAmount)*

Function parameters:

1. providingTokenRoot - the root address of the TIP-3 token that will be used to enter liquidity;
2. providingTokenAmount - the number of tokens to be entered;

Return value:

1. anotherTokenAmount - the number of tokens of another TIP-3 token that must be entered;

Description of the function: this function allows you to calculate how many tokens must be provided if the providingTokenAmount is provided by one of the two tokens.

- *createSwapPayload (address sendTokensTo) external pure returns (TvmCell)*

Function parameters:

1. sendTokensTo - the address of the user's wallet to which tokens will be received after the exchange;

Returned value: payload, which must be attached to perform the exchange operation.
Description of the function: this function is used to get the payload required to perform the exchange operation.

- *createProvideLiquidityPayload (address tip3Address) external pure returns (TvmCell)*

Function parameters:

1. tip3Address - the address of the user's LP-token wallet, if the user does not have an LP-wallet, the address must be zero;

Returned value: payload, which must be attached to perform the liquidity input operation for two tokens.

Description of the function: this function is used to get the payload required to perform the operation of entering liquidity for two tokens.

- *createProvideLiquidityOneTokenPayload (address tip3Address) external pure returns (TvmCell)*

Function parameters:

1. tip3Address - the address of the user's LP-token wallet, if the user does not have an LP-wallet, the address must be zero;

Returned value: payload, which must be attached to perform the operation of entering liquidity for one token;

Description of the function: this function is used to receive the payload required to perform the operation of entering liquidity for one token.

- *createWithdrawLiquidityPayload (address tokenRoot1, address tokenWallet1, address tokenRoot2, address tokenWallet2) external pure returns (TvmCell)*

Function parameters:

1. tokenRoot1 - the address of the TIP-3 root contract of the wallet that owns the tokenWallet1 wallet;
2. tokenWallet1 - TIP-3 address of the user's wallet;
3. tokenRoot2 - the address of the TIP-3 root contract of the wallet that owns the tokenWallet2 wallet;
4. tokenWallet2 - TIP-3 address of the user's wallet;

Returned value: payload, which must be attached to perform the operation of withdrawing liquidity for two tokens;

Description of the function: this function is used to receive the payload required to perform the operation of withdrawing liquidity for two tokens.

- *createWithdrawLiquidityOneTokenPayload (address tokenRoot, address userWallet) external pure returns (TvmCell)*

Function parameters:

1. tokenRoot - the address of the TIP-3 root contract of the wallet that owns the userWallet wallet;
2. userWallet - TIP-3 address of the user's wallet;

Returned value: payload, which must be attached to perform the operation of withdrawing liquidity for one token;

Description of the function: this function is used to receive the payload required to perform the operation of withdrawing liquidity for one token.

IUpgradeSwapPairCode interface

- *checkIfSwapPairUpgradeRequired (uint32 newCodeVersion) external returns (bool)*

Function parameters:
newCodeVersion - swap-pair code version;
Returned value: whether the swap pair needs to be updated;

Function Description: This function checks if swap pair update is required based on swap pair code version check;

- *updateSwapPairCode (TvmCell newCode, uint32 newCodeVersion) external*

Function parameters:

1. newCode - new swap pair code;
2. newCodeVersion - new version of the swap-pair code;

Function description: this function updates the swap pair code;

Description of events generated by a swap pair

- Swap (address providedTokenRoot, address targetTokenRoot, uint128 tokensUsedForSwap, uint128 tokensReceived, uint128 fee) - created upon successful token exchange operation;
- ProvideLiquidity (uint256 liquidityTokensAmount, uint128 firstTokenAmount, uint128 secondTokenAmount) - created upon successful liquidity input operation;
- WithdrawLiquidity (uint256 liquidityTokensAmount, uint128 firstTokenAmount, uint128 secondTokenAmount) - created upon a successful liquidity withdrawal operation;
- UpdateSwapPairCode (uint32 newCodeVersion) - created when the contract code update operation is successful.

Description of the system of interaction between smart contracts

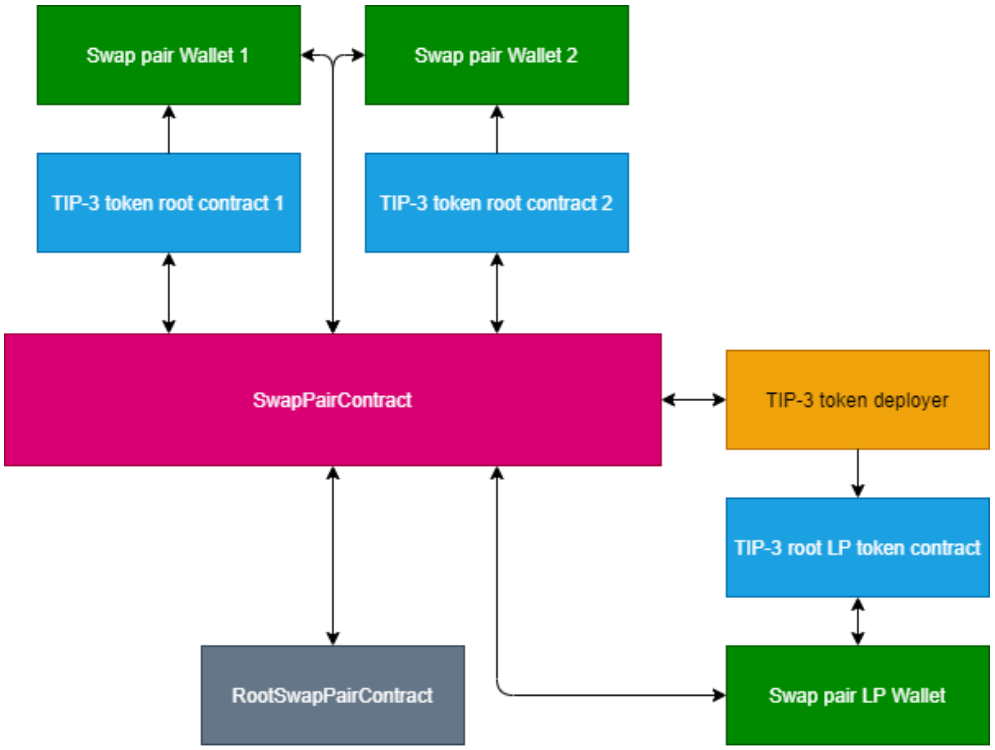
Description of SwapPairContract smart contract system

Developed solution consists of several smart contracts interacting with each other. Used smart contracts:

1. RootSwapPairContract - root contract of swap pairs. It is responsible for creation of new swap pairs and is controlling uniqueness of swap pairs created by this RootSwapPairContract;
2. SwapPairContract - smart contract realizing liquidity pool mechanic;
3. TIP3TokenDeployer - smart contract used for creation of new TIP-3 tokens;
4. TONTokenRoot - root smart contract of TIP-3 tokens;
5. TONTokenWallet - TIP-3 token wallet smart contract;

Smart contracts that interact with SwapPairContract:

1. Root TIP-3 token smart contracts for which swap pair is created and LP token root smart contract;
2. Smart contracts for TIP-3 token wallets for which a swap pair has been created, smart contract for a TIP-3 LP token wallet;
3. Smart contract, that creates new swap pairs;
4. Root swap pair contract;



Smart contracts interacting with swap pair

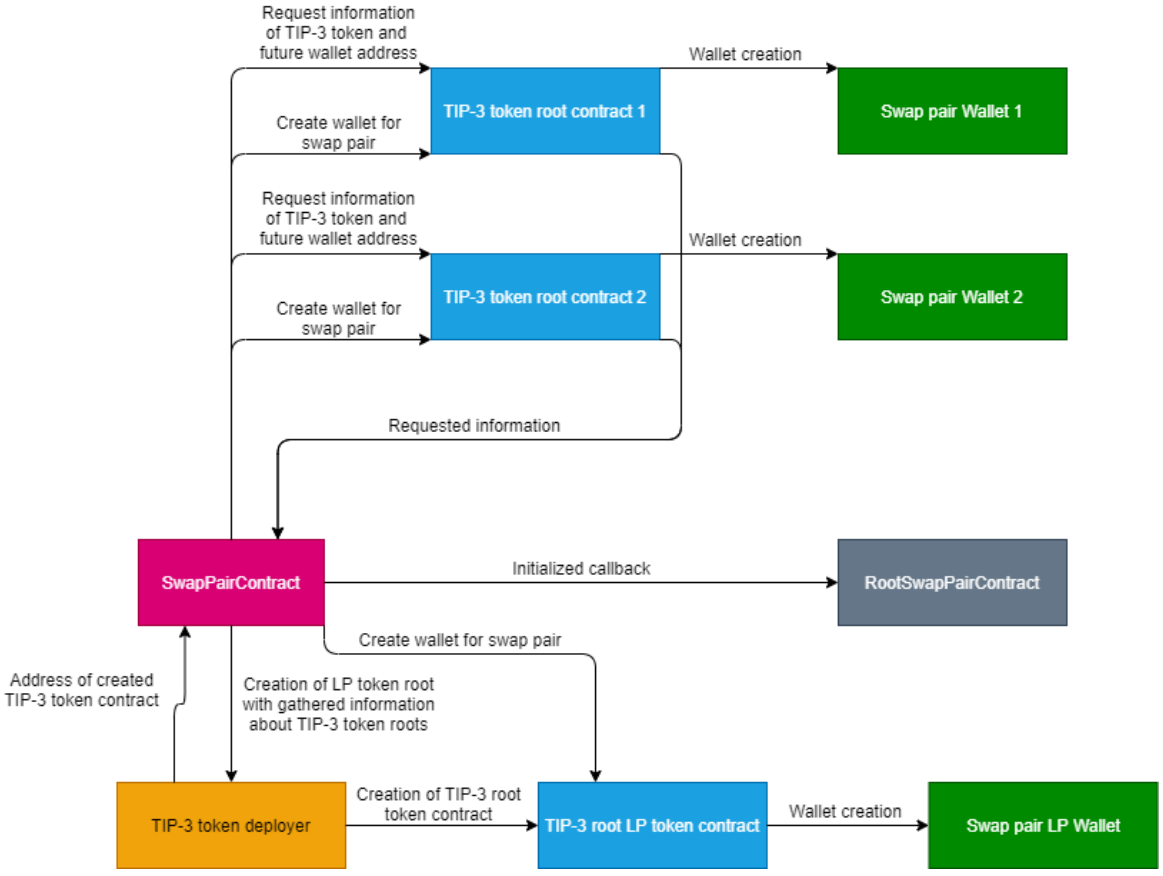
Description of swap pair creation process:

1. User calls function deploySwapPair of SwapPairRootContract and specifies addresses of TIP-3 token root smart contracts - tokenRoot1 and tokenRoot2;
2. Smart-contract RootSwapPairContract creates a record of new swap pair and deploys new swap pair contract

Creation of swap pair

Description of swap pair initialization process:

1. Swap pair fetches information about TIP-3 tokens - tokenRoot1 and tokenRoot2;
2. LP token root smart contract is created using TIP-3TokenDeployer smart contract;
3. TIP-3 token wallets are created - wallets for tokens tokenRoot1, tokenRoot2 and wallet for LP tokens;
4. RootSwapPairContract callback is called to signal that swap pair is fully initialized and ready to go.



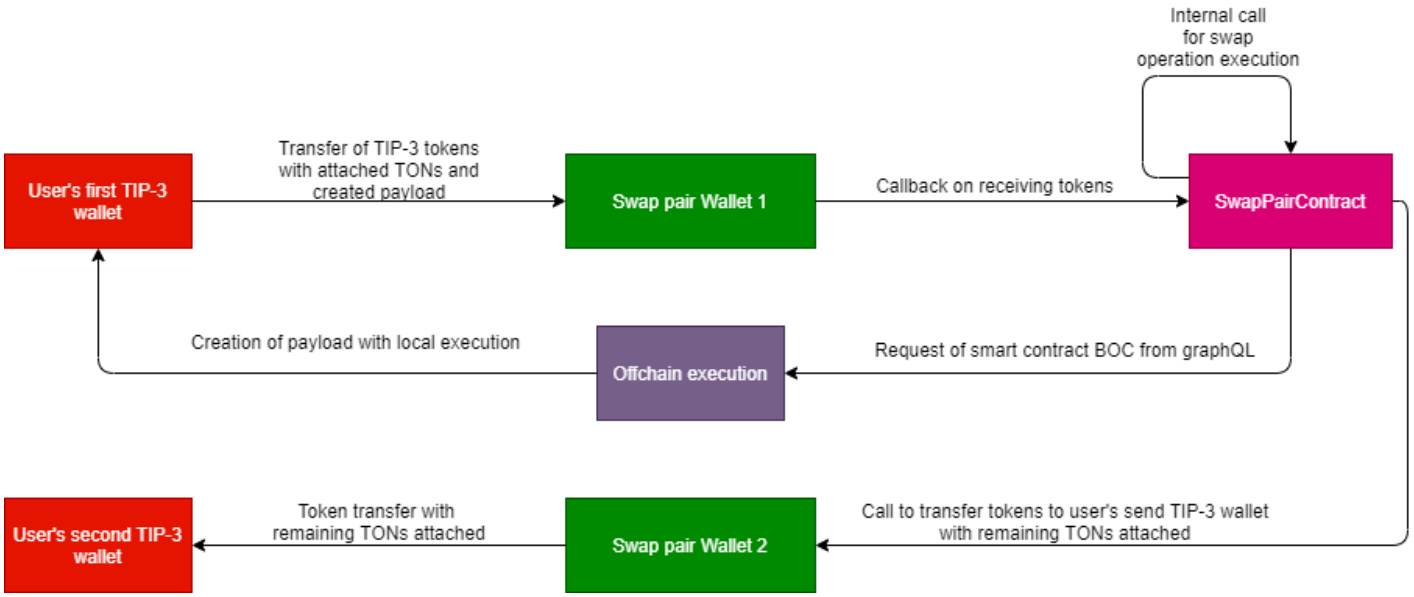
Initialization of swap pair

Approximate time of swap pair initialization from the deploySwapPair call to initialized callback call is ~30-60 seconds.

Description of swap pair operations execution

Description of swap operation execution:

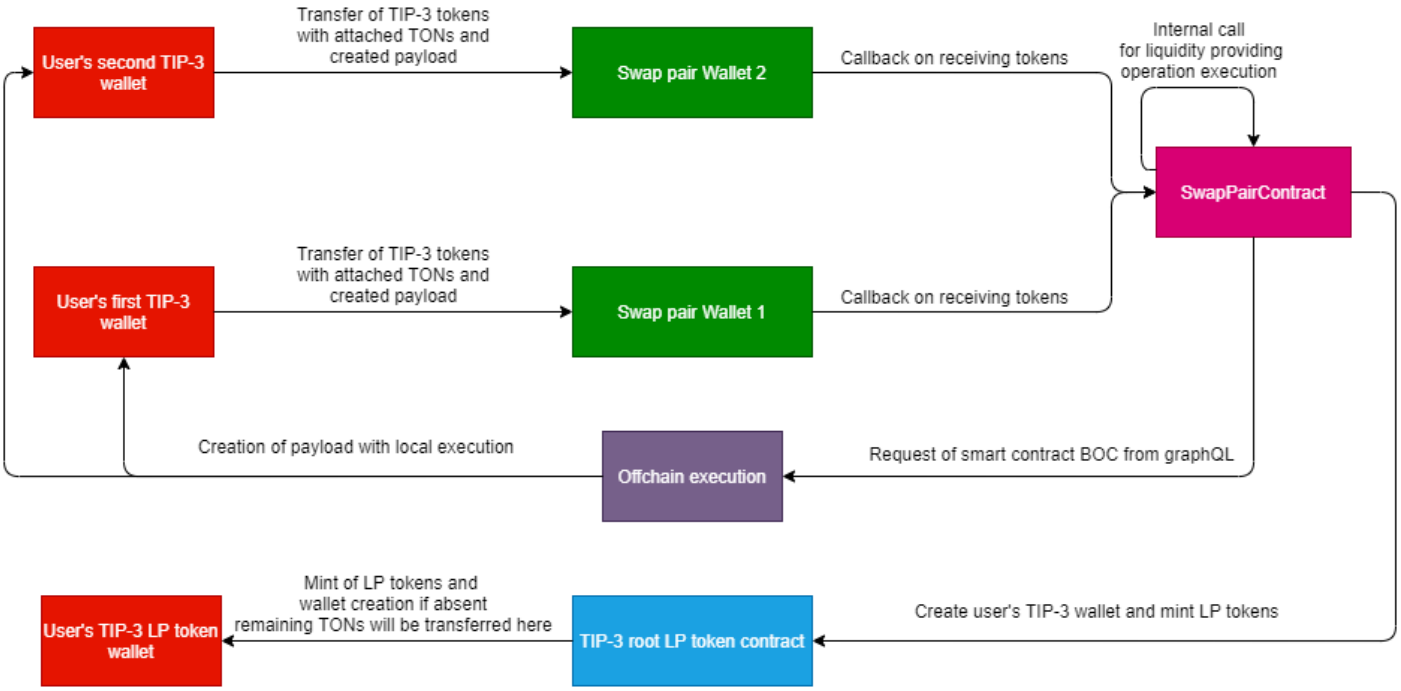
1. User transfers TIP-3 tokens to corresponding swap pair's wallet with created payload (by calling SwapPairContract funcion offchain or in debot) and attached TONs;
2. Swap pair's TIP-3 wallet calls swap pair's callback and transfers payload and information required to perform swap operation;
3. Swap operation is executed;
4. If operation finishes successfully than swapped tokens are transferred to TIP-3 wallet specified in previously created payload, if operation fails, than tokens are transferred back to sender.



Successful swap pair operation execution

Description of providing liquidity to swap pair using two tokens:

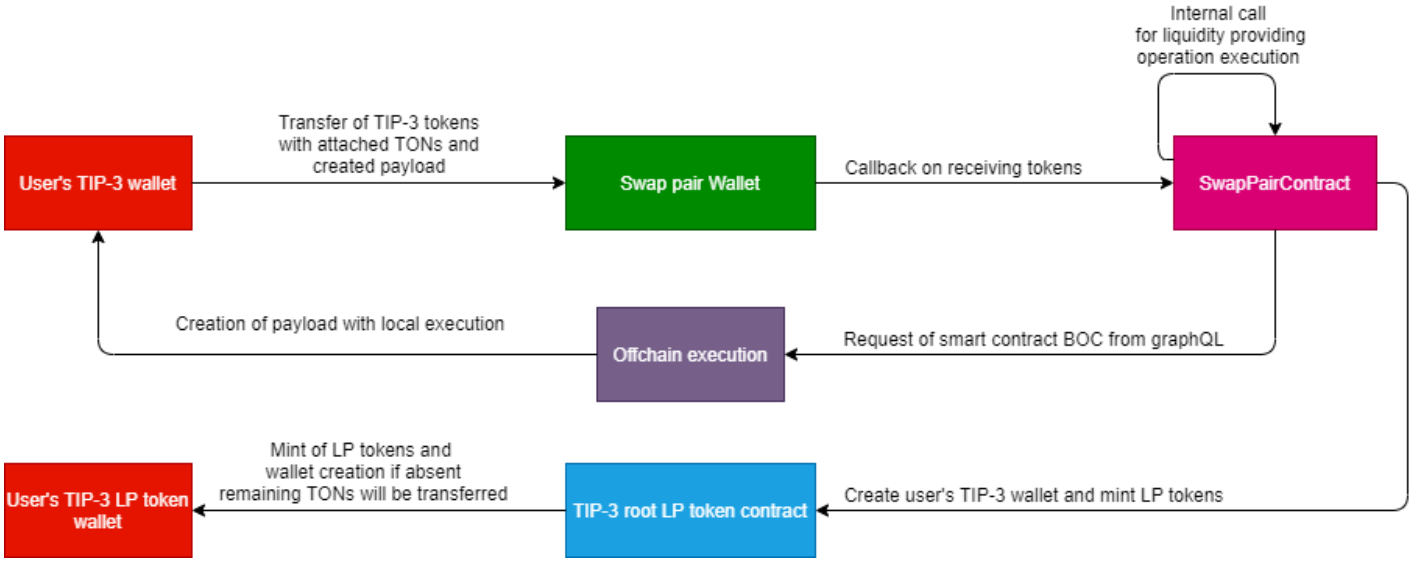
1. User transfers tokens from the first TIP-3 wallet to corresponding swap pair TIP-3 wallet with attached TONs and payload;
2. Swap pair's wallet which receives tokens calls swap pair's callback function and transfers information required to perform the first stage of liquidity providing operation;
3. Swap pair registers information;
4. User transfers tokens from the second TIP-3 wallet to corresponding swap pair TIP-3 wallet with attached TONs and payload;
5. Swap pair's wallet which receives tokens calls swap pair's callback function and transfers information required to finish liquidity providing operation;
6. The swap pair registers information that the second root tokens have been transferred;
7. The process of providing liquidity is performed, during which LP tokens are minted to the address specified when creating the payload, if the user does not have an LP wallet, then a wallet for LP tokens will be created with the same parameters as the user's second TIP-3 wallet;



Successful liquidity providing operation execution

Description of the process of providing liquidity via one token:

1. The user transfers tokens from the TIP-3 wallet to the corresponding swap pair's wallet with the attached TONs and a previously created payload;
2. The swap pair's wallet to which the tokens were transferred calls the swap pair callback and transfers the information necessary to perform the liquidity providing by one token;
3. Liquidity providing operation via one token is performed; ;
4. If the operation is successful, LP tokens are minted to the user's LP wallet specified in the payload, if the wallet does not exist, then it will be created.

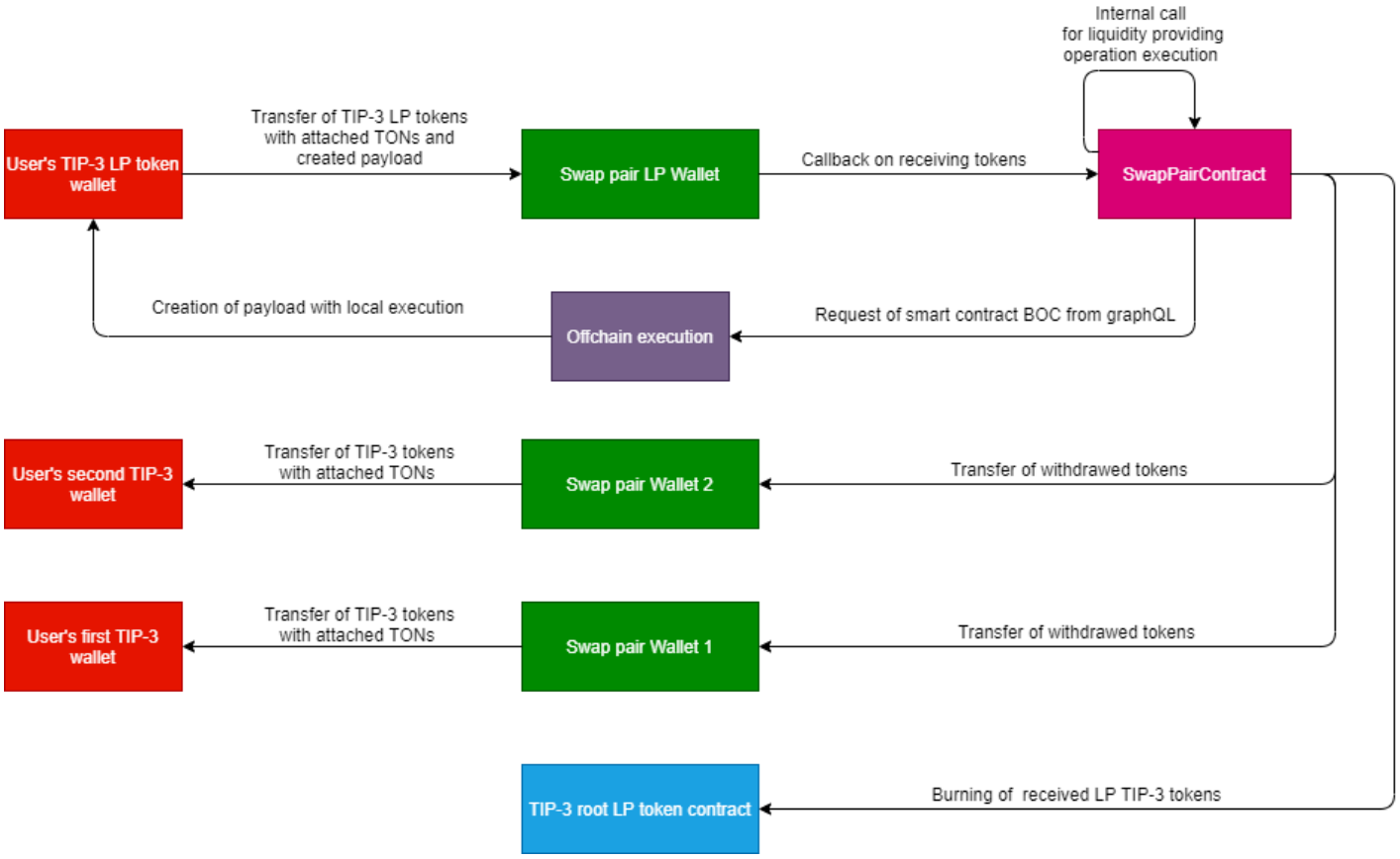


Successful execution of an operation for providing liquidity for one token

Description of the liquidity withdrawal process via two tokens:

1. The user transfers tokens from the TIP-3 LP wallet to the corresponding swap pair's wallet with attached TONs and a pre-created payload;
2. The swap pair's wallet to which the tokens were transferred calls the swap-pair callback and transfers the information necessary to perform the withdrawal of liquidity via two tokens;

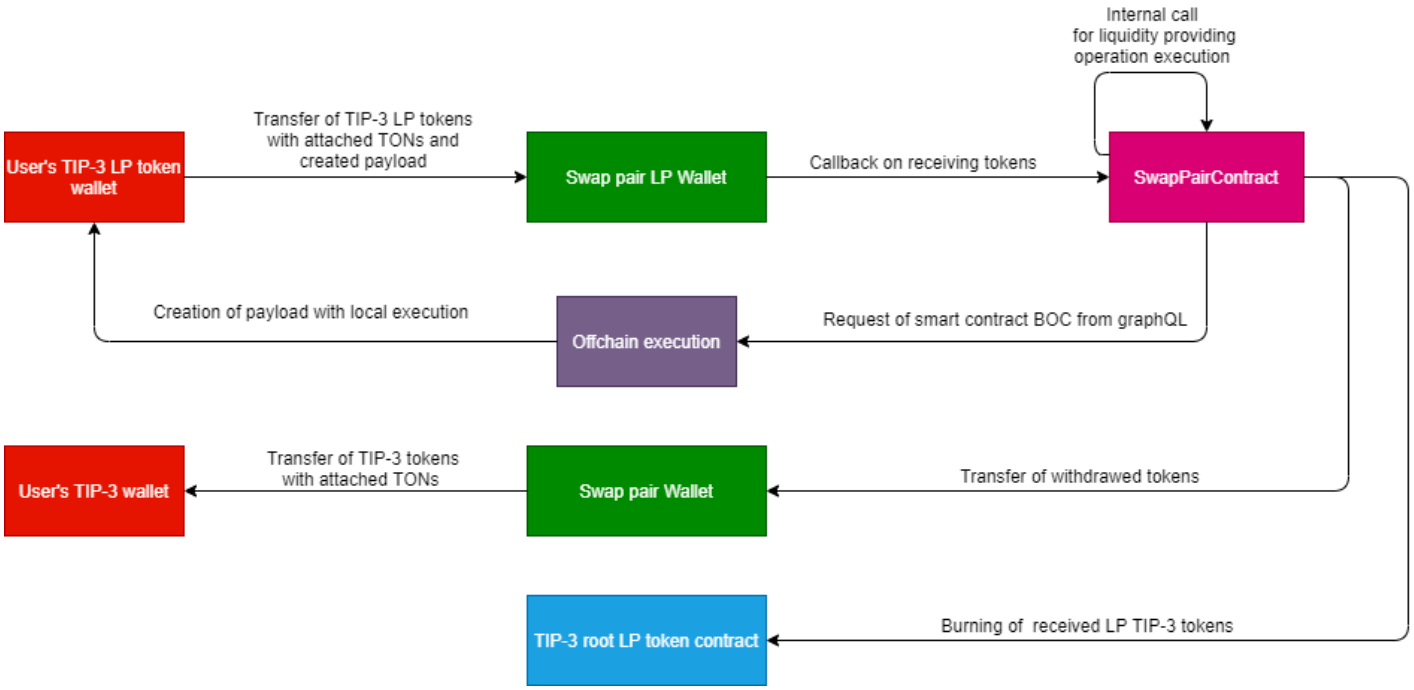
3. The operation of withdrawing liquidity from the swap pair is performed;
4. If the operation is successful, the withdrawn tokens are transferred to the wallets specified in the created payload, otherwise, the tokens are transferred back to the user's TIP-3 LP wallet;
5. Received LP tokens are burned.



Successful execution of liquidity withdrawal operation via two tokens

Description of liquidity withdrawing via one token operation:

1. User transfers tokens from TIP-3 LP wallet to the corresponding swap pair wallet with created payload and attached TONs;
2. Swap pair's wallet calls callback function of swap pair and transfers information required to perform liquidity withdrawing via one token operation;
3. Swap pair executes liquidity withdrawing via one token operation;
4. If operation finishes successfully then withdrawn tokens are transferred to the wallet that was specified during payload creation. If operation fails, LP tokens would be transferred back to the user's LP wallet;
5. Received LP tokens are burnt;



Successful liquidity withdrawing operation via one token

Amount of TON recommended to attach to message for performing operations

- For swap operation: 0.2 TON;
- For liquidity providing operation via two tokens:
    - For the first transfer: 0.2 TON;
    - For the second transfer: 0.2 TON if LP wallet does not exist, 0.4 TON if LP wallet exists;
- For liquidity providing operation via one token: 0.2 TON if LP wallet exists, 0.6 TON if LP wallet does not exist;
- For liquidity withdrawing operation via two tokens: 0.8 TON;
- For liquidity withdrawing operation via one token: 0.5 TON.

Developed debots

Debot-explorer for swap pairs

This debot allows you to perform the following operations:

- Acquiring list of existing swap pairs;
- Acquiring information about swap pair - pool volumes, exchange rate;
- Creation of new swap pairs.

Debots for interactions with swap pairs

To interact with swap pair following debots were developed:

- Debot for executing swap operation;
- Debot for executing liquidity providing operation;
- Debot for executing liquidity providing via one token operation;
- Debot for executing liquidity withdrawing operation;
- Debot for executing liquidity withdrawing via one token operation.

СОДЕРЖАНИЕ:

ПРЕДСТАВЛЕНИЕ

Всем привет!
Мы команда SVOI.dev и в представленном проекте - результаты нашего длительного кропотливого труда, осуществляемого в рамках контеста, проводимого FreeTON Community:
#23 FreeTon DEX Implementation Stage 2.
Целью данной работы стала разработка и имплементация быстрого обмена (Swap) в блокчейне FreeTON и инфраструктуры, базирующегося на основе подхода “пул ликвидности”.

Ссылка на сайт проекта: https://swap.tonswap.com
Ссылка на обозреватель: https://explorer.tonswap.com
GitHub:
https://github.com/SVOIcom/tonswap-SC
https://github.com/SVOIcom/ton-testing-suite
https://github.com/SVOIcom/tonswap-frontend
https://github.com/SVOIcom/tonswap-explorer
https://github.com/SVOIcom/debots-collection

Кошелек: 0:0439186aa0147661ebaf2b32ecc76bac172fcdaa24c7df7c9cb03cc816e435e6

Контактная информация

Telegram (корпоративный акк): https://t.me/svoidev
сайт команды: https://SVOI.dev

Базовая экономическая модель и описание денежного потока

В результате реализации нашей командой представленного проекта - TonSwap - была создана архитектура имплементации быстрого обмена (Swap) в блокчейне FreeTON и инфраструктура базирующиеся на основе общей концепции:
Automated Market Maker (AMM) с использованием формулы константного продукта.

Автоматизированные маркет-мейкеры являются основой пространства DeFi. Они позволяют практически любому легко и эффективно создавать рынки. Хотя у них есть свои ограничения по сравнению с биржами заказов, общие инновации, которые они привносят в криптовалюту, неоценимы.

Биржи AMM превосходят концепцию книги заказов. Вместо того, чтобы устанавливать цены на основе спроса и предложения, AMM объединяет ликвидность и устанавливает цены с помощью детерминированной формулы ценообразования.

Считаем, что именно данное решение в части DEX архитектуры - наиболее разумное, выгодное и эффективное для использования в сети TOH.

Давайте посмотрим с экономической точки зрения, как такая модель может побудить пользователей использовать ее для своих повседневных операций обмена и в то же время выиграть от роста сети в целом и обмена в частности.

Основная мотивация очевидна: люди хотят переносить свои активы в блокчейне в зависимости от своих потребностей и целей, они хотят обменять актив и в лучшем случае получить доход за предоставление ликвидности.

С другой стороны, хорошая реализация DEX должна обеспечить стабильность процесса обмена, принести некоторые дивиденды для активных пользователей и поставщиков ликвидности и предоставить некоторый механизм управления для будущих изменений и улучшений.

AMM (для простых пользователей) можно представить как робота, который всегда готов предложить цену для обоих активов.

С помощью AMM можно безопасно торговать, а также самому становиться хостом пар / пулов, что позволяет практически любому стать маркет-мейкером на бирже и получать комиссию за предоставление ликвидности.

Простым языком модель функционирования TonSwap выглядит следующим образом:

1. поставщики ликвидности размещают пару токенов - в существующие пулы ликвидности или образовывая новые;

- в случае создания новой пары - первичный поставщик ликвидности задаёт курс обмена одного токена из пары на другой;
- в случае, когда поставщик ликвидности вносит токены в уже существующую пару (образованный пул ликвидности) - токены должны вносится в пул в соответствии с обменным курсом между токенами пары, существующим на момент внесения токенов в пул;

2. при размещении ликвидности (пары токенов) поставщики получают виртуальный “токен” пары (Определяется как отношение`вложенная пользователем ликвидность` / `общая ликвидность`) - в подтверждение доли участия в пуле ликвидности;

3. трейдеры / покупатели через созданный Swap могут обменять имеющиеся у них токены на другие токены в рамках существующих пар по курсу, рассчитанному на основании объема токенов в свап паре / пуле ликвидности на момент осуществления операции;

4. при осуществлении обмена трейдером уплачивается комиссия, сумма которой распределяется между поставщиками ликвидности пула, в котором произошёл обмен.

Распределение комиссии происходит в соответствии с долями поставщиков ликвидности.

Награды определяются протоколом. В настоящее время комиссия составляет 0,3%, по примеру Uniswap v2, в последующем размер комиссии можно изменить. Другие платформы или форки могут взимать меньшую плату, чтобы привлечь больше поставщиков ликвидности в свой пул.

5. при осуществлении обмена одного токена на другой в рамках определённого пула - изменяется их соотношение внутри пула в зависимости от объёма выведенных и внесённых токенов;

*осуществление транзакции обмена возможно в случае, когда у покупателя токенов в кошельке достаточный объём другого вида токенов пары для обмена, а также для покрытия комиссии, уплачиваемой поставщикам ликвидности;

6. чтобы операции обмена не имели большого влияния на стоимость одного токена по отношению к другому - для Swap наиболее благоприятные условия функционирования, когда:

- существует большое количество поставщиков ликвидности и покупателей / трейдеров
- проведение операций обмена происходит в суммах небольших - относительного сформированного пула ликвидности пары.

Но это условия, на которые прямо повлиять нет возможности.

7. Важно описать ещё одно из действий, осуществляемых в рамках Swap Pair - выведение поставщиком ликвидности вложенных токенов:

если в определённый момент поставщик ликвидности принимает решение вывести свои токены из пула, то ему предоставляется объём токенов не равный первоначальному, а равный его доле от общего объёма.

Например:

Поставщик ликвидности N первоначально вкладывает в пул пару токенов - 1 WETH и 1 WBTC - исходя из установленного курса 1:1.

на момент внесения в пул уже вложено 9 WETH и 9 WBTC, т.е. его доля в пуле равна 10%.

Допустим в течение какого-то времени происходит многочисленный обмен одних токенов на другие в результате чего получаем следующее соотношение токенов пары:

6 WETH и 20 WBTC.

Если в период времени данного установленного соотношения поставщик ликвидности N принимает решение вывести свои токены из пула, то он получает их в объёме:

0.6 WETH и 2 WBTC.

Об эффективности автоматизированных маркет-мейкеров

В моделях AMM цена определяется по детерминированным формулам, основанным на спросе и предложении активов. В результате эти модели полагаются на арбитражеров, чтобы уравнять цену с другими рынками. Другими словами, когда есть разница в цене, арбитражёры начинают покупать / продавать активы из смарт-контракта, который изменяет спрос и предложение актива и выравнивает цену. Однако это приводит к неэффективности поставщиков ликвидности, поскольку они несут убытки (или альтернативные издержки по сравнению с реальной рыночной ценой, которую используют арбитражёры).

Все вышеперечисленное можно представить в виде базовой формулы:

У нас есть рынок, на котором можно обменять монеты типа α на монеты типа β (и наоборот). Этот рынок имеет резервы Rα> 0 и Rβ> 0, постоянный продукт k = RαRβ и процентную плату (1 - γ). Сделка на этом рынке, торгующая ∆β> 0 монет β для ∆α> 0 монет α, должна удовлетворять

$$(R_\alpha - \Delta_\alpha)(R_\beta + \gamma\Delta_\beta) = k.$$

После каждой транзакции резервы обновляются следующим образом: Rα "→ Rα - ∆α, Rβ" → Rβ + ∆β и k "→ (Rα - ∆α) (Rβ + ∆β). Мы всегда будем требовать что Rα, Rβ> 0, так что любая сделка, которая приводит к неположительному резерву, никогда не выполняется (эквивалентно, мы говорим, что такая сделка имеет бесконечную стоимость).

Когда комиссия равна нулю (т. Е. Г = 1), любая сделка ∆β на ∆α должна изменять резервы таким образом, чтобы продукт RαRβ оставался равным константе k.

Подробное техническое описание предлагаемой реализации с обоснованием выбранного подхода: смарт-контракты, уровень интеграции, интерфейсы

Разработанные Смарт-Контракты:

1. SwapPairRootContract;
2. SwapPairContract;
3. TIP3TokenDeployer.

Используемые вспомогательные контракты:

1. TONTokenWallet;
2. RootTokenContract.

Описание разработанных Смарт-Контрактов

SwapPairRootContract

Описание смарт-контракта

Данный смарт-контракт используется для учёта созданных свап-пар, а также для создания новых уникальных свап-пар.

Описание функций смарт-контракта

Функции данного смарт-контракта, которые представляют какой-либо интерес для разработчиков описаны в следующем разделе. Неописанные функции являются служебными и не предназначены для вызовов извне (модификаторы или внутренние функции).

Описание интерфейсов смарт-контракта

Интерфейс IRootSwapPairContract

- *deploySwapPair(address tokenRootContract1, address tokenRootContract2) external returns (address)*
  Параметры функции:
  1. tokenRootContract1 - адрес первого рут-контракта токена;
  2. tokenRootContract2 - адрес второго рут-контракта токена;

  Возвращаемые значения: адрес будущей свап-пары;
  Описание функции: данная функция используется для создания новых свап-пар. Для создания новой свап-пары требуется вместе с вызовом функции перевести требуемое количество TONов, которые будут использованы для инициализации свап-пары. Конкретное значение зависит от рут-контракта.
- *checkIfPairExists(address tokenRootContract1, address tokenRootContract2) external view returns (bool)*
  Параметры функции:
  1. tokenRootContract1 - адрес первого рут-контракта токена;
  2. tokenRootContract2 - адрес второго рут-контракта токена;

  Возвращаемые значения: существует свап-пара для данных токенов или нет.

Описание функции: проверяется, существует ли для заданной пары токенов свап-пара или нет;
- *getServiceInformation() external view returns (ServiceInfo)*
  Возвращаемые значения: сервисная информация о рут-контракте;
  Описание функции: получение сервисной информации о рут-контракте;
- *getAllSwapPairsID() external view returns (uint256[] ids):*
  Возвращаемые значения: массив из всех существующих ID свап-пар;
  Описание функции: данная функция возвращает уникальные ID всех созданных через данный рут-контракт свап-пар.
- *getPairInfoByID(uint256 uniqueID) external view returns (SwapPairInfo swapPairInfo)*
  Параметры функции:
  1. uniqueID - уникальный идентификатор функции;

  Возвращаемые значения: информация о свап-паре;

Описание функции: данная функция возвращает информацию о свап-паре по её уникальному ID. Так как процесс инициализации свап-пары занимает некоторое время, при запросе информации сразу после создания свап-пары, полученная информация окажется неполной. Необходимо подождать завершения инициализации свап-пары.

- *getPairInfo(address tokenRootContract1, address tokenRootContract2) external view returns (SwapPairInfo)*

Параметры функции:

1. tokenRootContract1 - адрес первого рут-контракта токена;
2. tokenRootContract2 - адрес второго рут-контракта токена;

Возвращаемые значения: информация о свап-паре;

Описание функции: аналогична функции getPairInfoByID, однако вместо использования уникального ID свап-пары используются адреса рут-контрактов TIP-3 токенов, для которых создавалась свап-пара.

- *setTIP3DeployerAddress(address tip3Deployer_) external*

  Параметры функции:

  1. tip3Deployer_ - адрес деплоера TIP-3 рут-контрактов;

  Описание функции: данная функция устанавливает адрес смарт-контракта, используемого для деплоя новых TIP-3 токенов. Данный контракт используется в процессе инициализации свап-пары для создания LP-токенов.
- *getFutureSwapPairAddress(address tokenRootContract1, address tokenRootContract2) external view returns(address)*

  Параметры функции:

  1. tokenRootContract1 - адрес первого рут-контракта токена;
  2. tokenRootContract2 - адрес второго рут-контракта токена;

  Возвращаемые значения: будущий адрес свап-пары;

  Описание функции: данная функция используется для получения будущего адреса свап-пары на основе переданных адресов рут-контрактов TIP-3 токенов.

Интерфейс IRootSwapPairUpgradablePairCode

- *setSwapPairCode(TvmCell code, uint32 codeVersion) external*

  Параметры функции:

  1. code - новый код свап-пары;
  2. codeVersion - версия кода смарт-контракта;

  Описание функции: данная функция используется для установки нового кода смарт-контракта, который будет использоваться для создания новых свап-пар. Также, новый код можно будет использовать для апгрейда уже существующих свап-пар.
- *upgradeSwapPair(address tokenRootContract1, address tokenRootContract2) external*

  Параметры функции:

  1. tokenRootContract1 - адрес первого рут-контракта токена;
  2. tokenRootContract2 - адрес второго рут-контракта токена;

  Описание функции: данная функция используется для апгрейда уже существующих свап-пар. Для запуска обновления необходимо приложить 3 TONa к отправляемому сообщению.

Интерфейс IRootContractCallback

- *swapPairInitializedCallback(SwapPairInfo spi) external*

  Параметры функции:

  1. spi - информация о свап-паре, полученная от самой свап-пары после процесса инициализации.

Описание функции: данная функция используется свап-парами для того, чтобы передать обновлённую информацию, содержащую адреса созданных кошельков и LP-токена. Может быть вызвана только свап-парой, созданной через данный рут-контракт.

Описание событий, создаваемых рут-контрактом свап-пары

- DeploySwapPair(address swapPairAddress, address tokenRootContract1, address tokenRootContract2) - данное событие создаётся при создании новой свап-пары;
- SwapPairInitialized(address swapPairAddress) - данное событие создаётся когда свап-пара завершает стадию инициализации и готова к функционированию.

TIP3TokenDeployer

Описание смарт-контракта

Данный смарт-контракт реализует механизм пар ликвидности.

Описание функций смарт-контракта

Функции данного смарт-контракта, которые представляют какой-либо интерес для разработчиков описаны в следующем разделе. Неописанные функции являются служебными и не предназначены для вызовов извне (модификаторы или внутренние функции). В данном случае у смарт-контракта отсутствуют прямой доступ к функциям свап-пары, таким как функции обмена, ввода и вывода ликвидности. Вызов данных функций осуществляется путём перевода на кошельки, принадлежащие смарт-контракту свап-пары с заранее созданным особым payload. Данный payload должен быть сформирован через вызов функций, принадлежащим свап-паре (они описаны ниже).

Описание интерфейсов смарт-контракта

Интерфейс ISwapPairContract

- *getExchangeRate(address swappableTokenRoot, uint128 swappableTokenAmount) external responsible view returns (SwapInfo _swapInfo)*
  Параметры функции:
  1. swappableTokenRoot - адрес смарт-контракта, токены которого будут предоставлены для операции обмена;
  2. swappableTokenAmount - количество токенов, предоставляемое для операции обмена токенов;

  Возвращаемое значение: информация об операции обмена токенов
  Описание функции: данная функция используется для получения информации о результате обмена, если обмен произойдёт в текущий момент;
- *getCurrentExchangeRate() external responsible view returns (LiquidityPoolsInfo lpi)*
  Возвращаемое значение: информация о текущем состоянии пулов ликвидности свап-пары;
  Описание функции: данная функция возвращает текущее состояние пулов ликвидности свап-пары;
- *getPairInfo() external responsible view returns (SwapPairInfo info)*
  Возвращаемое значение: информация о свап-паре;
  Описание функции: данная функция возвращает информацию о свап-паре;
- *getWithdrawingLiquditiyInfo(uint256 liquiditiyTokensAmount) external view returns (uint128 withdrawedFirstTokenAmount, uint128 withdrawedSecondTokenAmount)*

  Параметры функции:

1. liquidityTokensAmount - выводимое количество LP-токенов;

Возвращаемые значения:

1. withdrawedFirstTokenAmount - сколько TIP-3 токенов первого рут-контракта TIP-3 будет выведено;
2. withdrawedSecondTokenAmount - сколько TIP-3 токенов второго рут-контракта TIP-3 будет выведено;

Описание функции: сколько TIP-3 токенов будет получено при выводе liquidityTokensAmount LP-токенов из свап-пары в данный момент;

- *getProvidingLiquidityInfo(uint128 maxFirstTokenAmount, uint128 maxSecondTokenAmount) external view returns (uint128 providedFirstTokenAmount, uint128 providedSecondTokenAmount)*

  Параметры функции:

  1. maxFirstTokenAmount - сколько будет внесено токенов первого TIP-3 контракта;
  2. maxSecondTokenAmount - сколько будет внесено токенов второго TIP-3 контракта;

  Возвращаемые значения:

  1. providedFirstTokenAmount - сколько будут введено первых TIP-3 токенов, если вывод произойдёт в данный момент;
  2. providedSecondTokenAmount - сколько будут введено вторых TIP-3 токенов, если вывод произойдёт в данный момент;

  Описание функции: данная функция позволяет определить сколько TIP-3 токенов будет внесено, если ввод ликвидности произойдёт в данный момент.

- *getAmotherTokenProvidingAmount(address providingTokenRoot, uint128 providingTokenAmount) external view returns (uint128 anotherTokenAmount)*

  Параметры функции:

  1. providingTokenRoot - адрес рута TIP-3 токена, который будет использован для ввода ликвидности;
  2. providingTokenAmount - количество вводимых токенов;

  Возвращаемое значение:

  1. anotherTokenAmount - количество токенов другого TIP-3 токена, которое необходимо ввести;

  Описание функции: данная функция позволяет вычислить какое количество токенов необходимо предоставить, если предоставлено providingTokenAmount одного из двух токенов.

- *createSwapPayload(address sendTokensTo) external pure returns (TvmCell)*

  Параметры функции:

  1. sendTokensTo - адрес кошелька пользователя, на который поступят токены после обмена;

  Возвращаемое значение: payload, который необходимо приложить для выполнения операции обмена.

  Описание функции: данная функция используется для получения payload, необходимого для выполнения операции обмена.

- *createProvideLiquidityPayload(address tip3Address) external pure returns (TvmCell)*

  Параметры функции:

  1. tip3Address - адрес кошелька LP-токена пользователя, если у пользователя отсутствует LP-кошелёк, адрес должен быть нулевым;

  Возвращаемое значение: payload, который необходимо приложить для выполнения операции ввода ликвидности по двум токенам.

  Описание функции: данная функция используется для получения payload, необходимого для выполнения операции ввода ликвидности по двум токенам.

- *createProvideLiquidityOneTokenPayload(address tip3Address) external pure returns (TvmCell)*

  Параметры функции:

  1. tip3Address - адрес кошелька LP-токена пользователя, если у пользователя отсутствует LP-кошелёк, адрес должен быть нулевым;

Возвращаемое значение: payload, который необходимо приложить для выполнения операции ввода ликвидности по одному токену;

Описание функции: данная функция используется для получения payload, необходимого для выполнения операции ввода ликвидности по одному токену.

- *createWithdrawLiquidityPayload(address tokenRoot1, address tokenWallet1, address tokenRoot2, address tokenWallet2) external pure returns (TvmCell)*

Параметры функции:

1. tokenRoot1 - адрес рут-контракта TIP-3 кошелька, которому принадлежит кошелёк tokenWallet1;
2. tokenWallet1 - адрес TIP-3 кошелька пользователя;
3. tokenRoot2 - адрес рут-контракта TIP-3 кошелька, которому принадлежит кошелёк tokenWallet2;
4. tokenWallet2 - адрес TIP-3 кошелька пользователя;

Возвращаемое значение: payload, который необходимо приложить для выполнения операции вывода ликвидности по двум токенам;

Описание функции: данная функция используется для получения payload, необходимого для выполнения операции вывода ликвидности по двум токенам.

- *createWithdrawLiquidityOneTokenPayload(address tokenRoot, address userWallet) external pure returns (TvmCell)*

Параметры функции:

1. tokenRoot - адрес рут-контракта TIP-3 кошелька, которому принадлежит кошелёк userWallet;
2. userWallet - адрес TIP-3 кошелька пользователя;

Возвращаемое значение: payload, который необходимо приложить для выполнения операции вывода ликвидности по одному токену;

Описание функции: данная функция используется для получения payload, необходимого для выполнения операции вывода ликвидности по одному токену.

Интерфейс IUpgradeSwapPairCode

- *checkIfSwapPairUpgradeRequired(uint32 newCodeVersion) external returns (bool)*

Параметры функции:

1. newCodeVersion - версия кода свап-пары;

Возвращаемое значение: требуется ли обновление свап-пары;

Описание функции: данная функция проверяет, требуется ли обновление для свап-пары на основе проверки версии кода свап-пары;

- *updateSwapPairCode(TvmCell newCode, uint32 newCodeVersion) external*

Параметры функции:

3. newCode - новый код свап-пары;
4. newCodeVersion - новая версия кода свап-пары;

Описание функции: данная функция осуществляет обновление кода свап-пары;

Описание событий, создаваемых свап-парой

5. Swap(address providedTokenRoot, address targetTokenRoot, uint128 tokensUsedForSwap, uint128 tokensReceived, uint128 fee) - создаётся при успешной операции обмена токенов;
6. ProvideLiquidity(uint256 liquidityTokensAmount, uint128 firstTokenAmount, uint128 secondTokenAmount) - создаётся при успешной операции ввода ликвидности;
7. WithdrawLiquidity(uint256 liquidityTokensAmount, uint128 firstTokenAmount, uint128 secondTokenAmount) - создаётся при успешной операции вывода ликвидности;

8. UpdateSwapPairCode(uint32 newCodeVersion) - создаётся при успешной операции обновления кода контракта.

Описание системы взаимодействия между смарт-контрактами

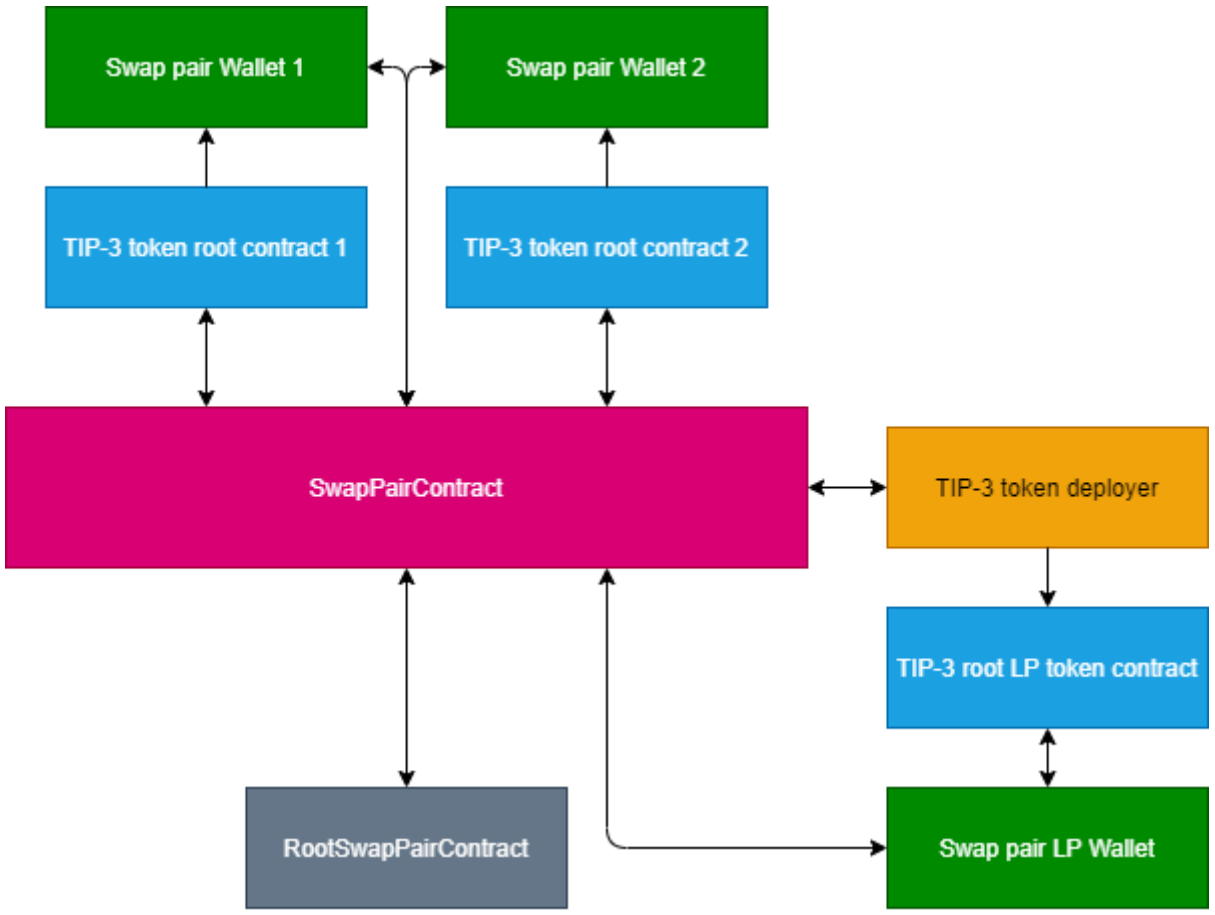
Описание системы смарт-контракта SwapPairContract

Разработанное решение состоит из нескольких взаимодействующих между собой смарт-контрактов. Используемые смарт-контракты:

1. RootSwapPairContract - рут-контракт свап-пар, ответственный за создание новых свап пар и контроль уникальности свап-пар, создаваемых данным рут-контрактом;
2. SwapPairContract - собственно смарт-контракт свап-пары;
3. TIP3TokenDeployer - смарт-контракт, используемый для создания новых TIP-3 токенов;
4. TONTokenRoot - рут-контракт TIP-3 токена;
5. TONTokenWallet - смарт-контракт кошелёк TIP-3 токена.

Смарт-контракты, взаимодействующие со смарт-контрактом свап-пары:

1. Смарт-контракты рутов TIP-3 токенов, для которых создана свап-пара, смарт-контракт рута TIP-3 LP токена;
2. Смарт-контракты кошельков TIP-3 токенов, для которых создана свап-пара, смарт-контракт кошелька TIP-3 LP токена;
3. Смарт-контракт, выполняющий создание TIP-3 LP токена;
4. Смарт-контракт рута свап-пары;



Смарт-контракты, взаимодействующие со свап-парой

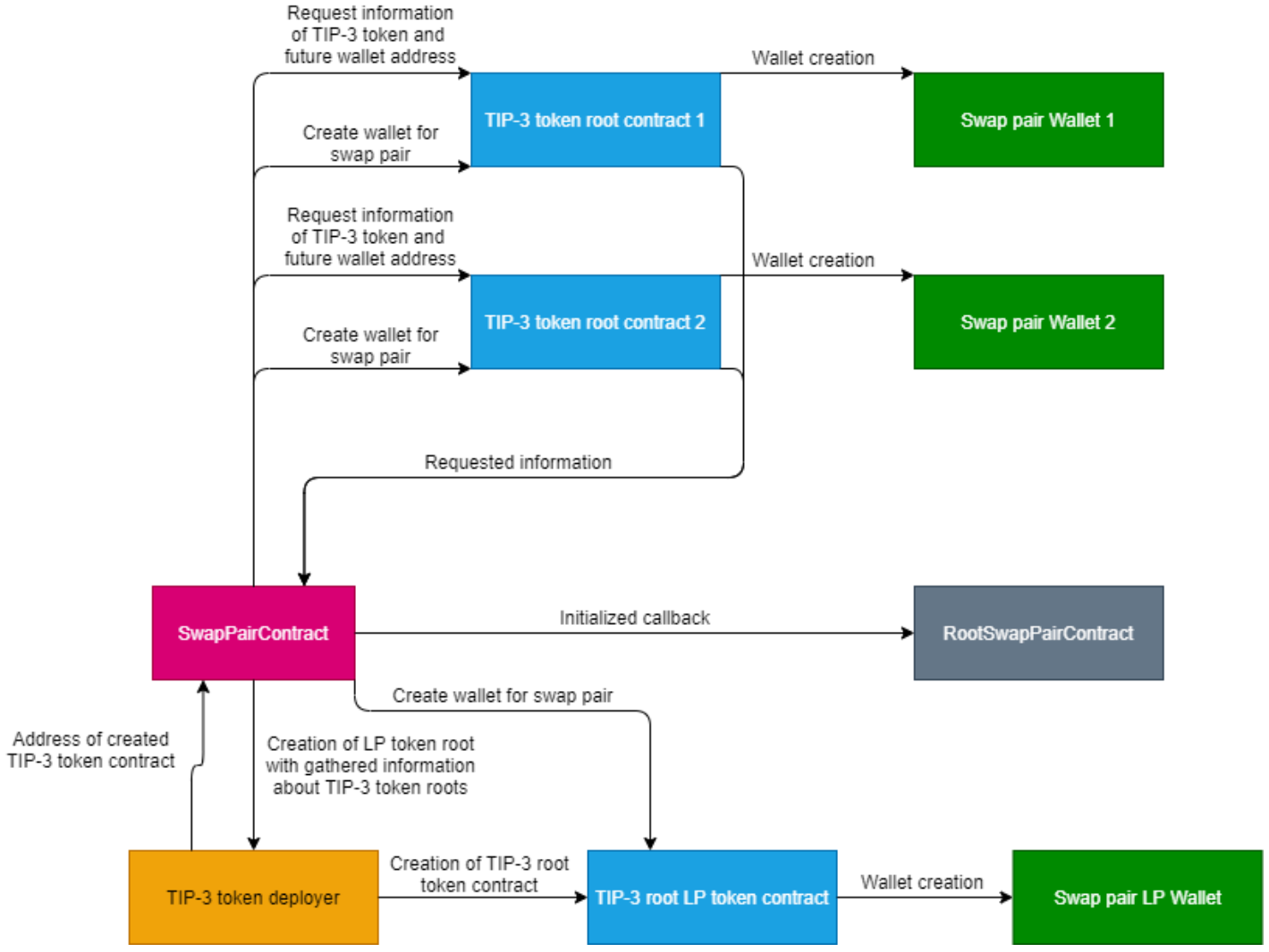
Описание создания контракта свап-пары:

1. Пользователь вызывает функцию deploySwapPair с указанием адресов рутов TIP-3 токенов tokenRoot1, tokenRoot2;
2. Смарт-контракт RootSwapPairContract создаёт запись о новой свап-паре и выполняет деплой новой свап-пары;

Создание свап-пары

Описание процесса инициализации свап-пары:

1. Происходит запрос информации о TIP-3 токенах tokenRoot1 и tokenRoot2;
2. Создаётся смарт-контракт рута LP-токена через смарт-контракт TIP3TokenDeployer;
3. Происходит создание кошельков для свап-пары - для токенов tokenRoot1, tokenRoot2 и для LP токена;
4. Вызывается коллбэк смарт-контракта RootSwapPairContract, сигнализирующий о том что свап-пара готова к работе;



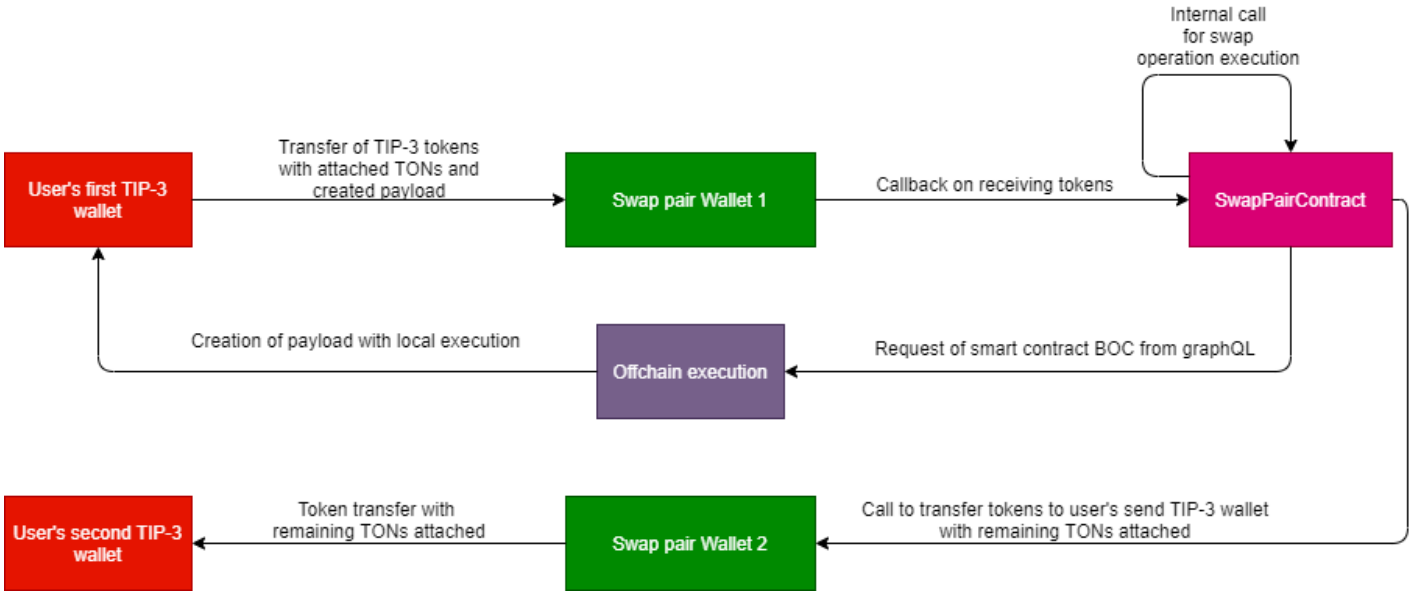
Инициализация свап-пары

Примерное время инициализации свап-пары от момента вызова смарт-контракта RootSwapPairContract до вызова его колбэка составляет примерно 30 секунд - 1 минуту.

Описание выполнения операций свап-пары

Описание процесса выполнения операции обмена:

1. Пользователь производит перевод TIP-3 токенов на кошелёк свап-пары с прикреплёнными тонами и предварительно созданным payload, информирующим пару о том, что требуется произвести операцию обмена;
2. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для выполнения обмена свап-паре;
3. Выполняется операция обмена;
4. Если операция выполнена успешно, то токены переводятся на TIP-3 кошелёк пользователя, указанный в ранее созданном payload, если операция не была выполнена - например, в паре нет ликвидности на данный момент или было переведено слишком мало TIP-3 токенов, токены отправляются обратно.

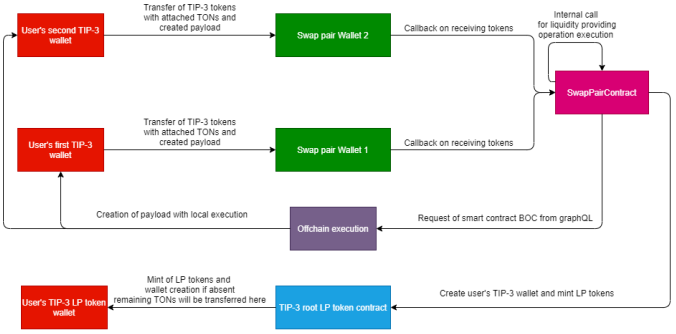


Успешное выполнение операции свапа

Описание процесса выполнения операции ввода ликвидности по двум токенам:

1. Пользователь переводит токены с первого TIP-3 кошелька на соответствующий кошелёк свап-пары с приложенными TOH-ами и предварительно созданным payload;
2. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для выполнения первого этапа ввода ликвидности;
3. Свап-пара регистрирует информацию о том, что выполнен первый этап ввода ликвидности по двум токенам;
4. Пользователь переводит токены со второго TIP-3 кошелька на соответствующий кошелёк свап-пары с приложенными TOH-ами и предварительно созданным payload;
5. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для завершения операции ввода ликвидности;
6. Свап-пара регистрирует информацию о том, что выполнен перевод токенов второго рута;

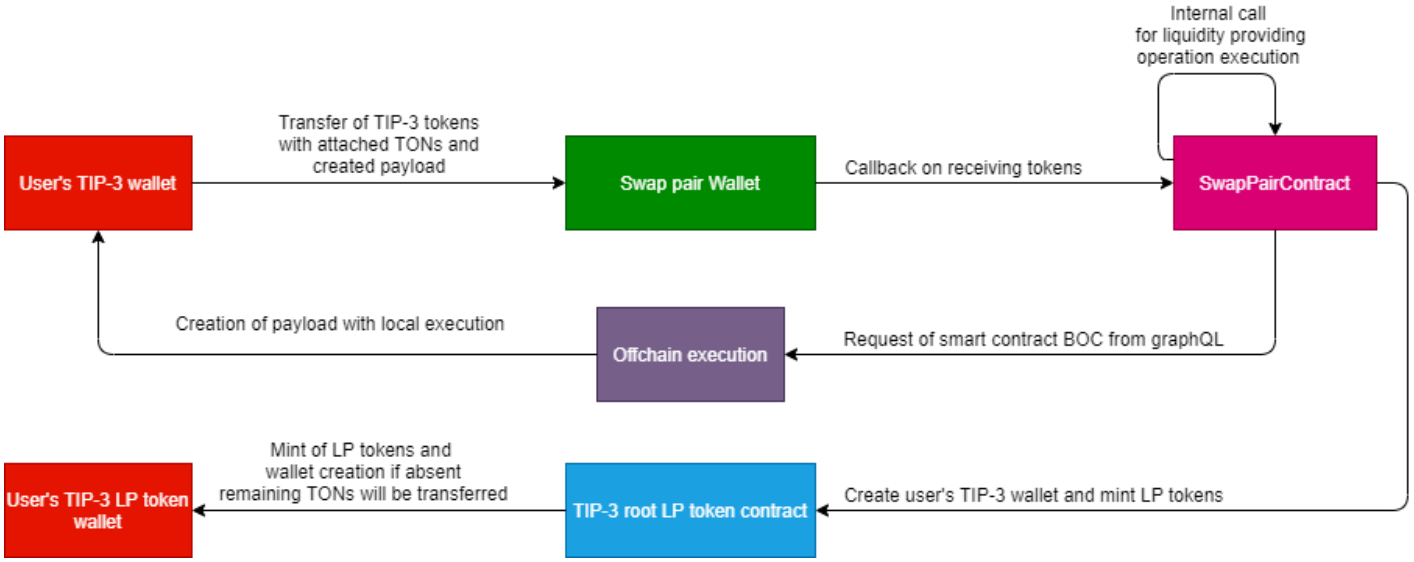
7. Выполняется процесс ввода ликвидности в ходе которого минтятся LP токены на адрес, указанный при создании payload, если LP-кошелька у пользователя нет, то будет создан кошелёк для LP токенов с теми же параметрами, что и второй TIP-3 кошелёк пользователя;



Успешное выполнение операции ввода ликвидности

Описание процесса ввода ликвидности по одному токену:

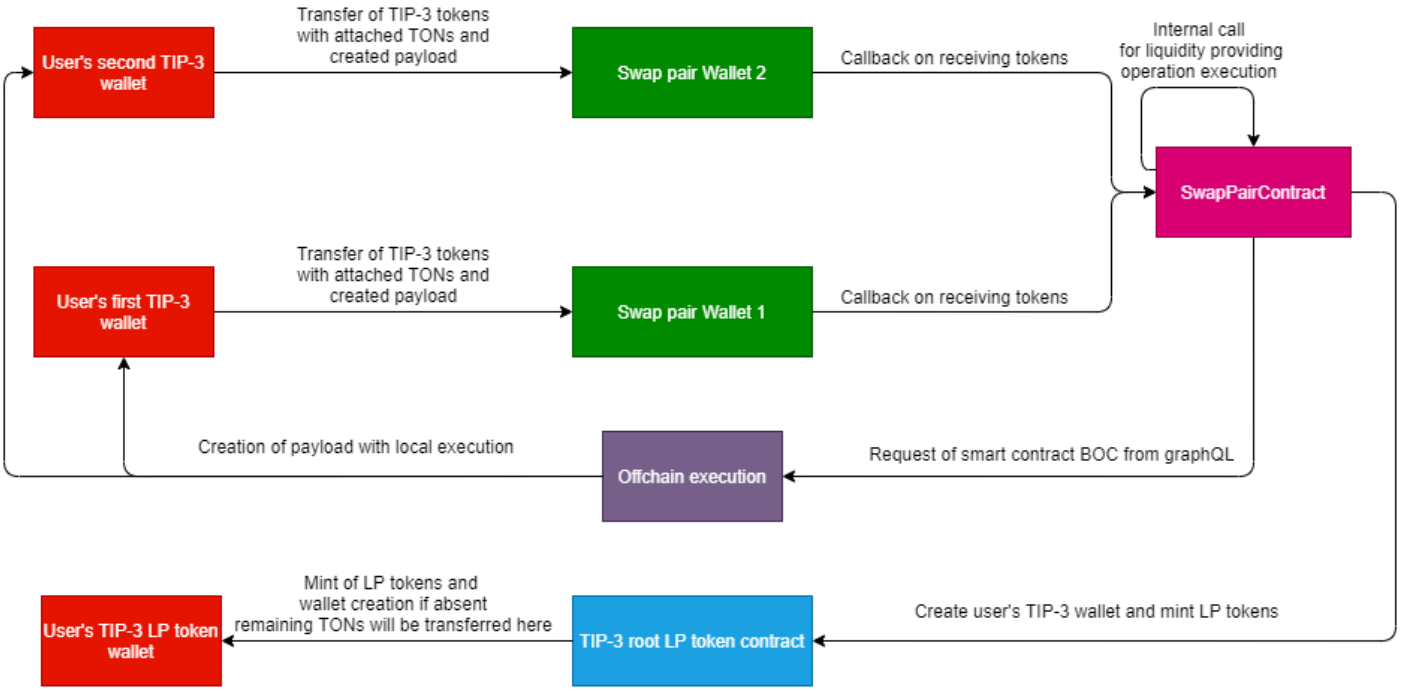
1. Пользователь переводит токены с TIP-3 кошелька на соответствующий кошелёк свап-пары с приложенными TOH-ами и предварительно созданным payload;
2. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для выполнения ввода ликвидности по одному токену;
3. Выполняется операция ввода ликвидности по одному токену;
4. Если операция выполнена успешно, LP токены минтятся на LP кошелёк пользователя, указанный в payload, если же кошелька не существует, то он будет создан.



Успешное выполнение операции ввода ликвидности по одному токену

Описание процесса вывода ликвидности по двум токенам:

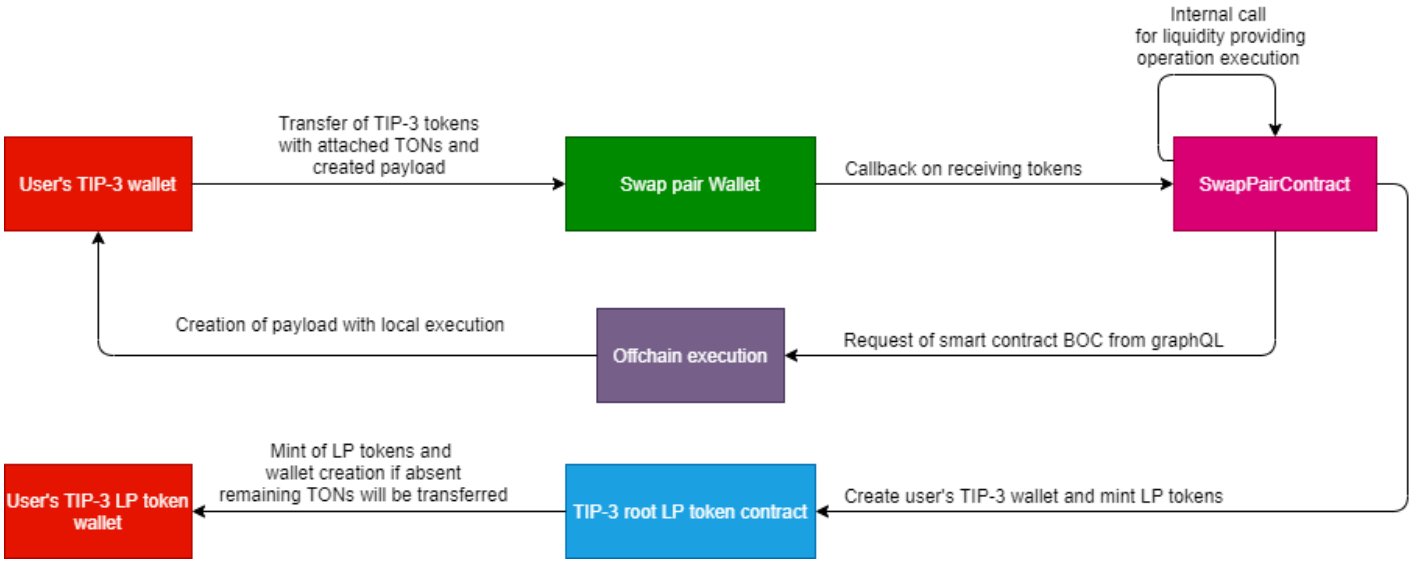
1. Пользователь переводит токены с TIP-3 LP кошелька на соответствующий кошелёк свап-пары с приложенными TOH-ами и предварительно созданным payload;
2. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для выполнения вывода ликвидности по двум токенам;
3. Выполняется операция вывода ликвидности из свап-пары;
4. Если операция выполнена успешно, то выведенные токены переводятся на кошельки, указанные в созданном payload, в ином случае, токены возвращаются на TIP-3 LP кошелёк пользователя;
5. Полученные LP токены сжигаются.



Успешное выполнение операции вывода ликвидности по двум токенам

Описание процесса вывода ликвидности по одному токену:

1. Пользователь переводит токены с TIP-3 LP кошелька на соответствующий кошелёк свап-пары с приложенными TOH-ами и предварительно созданным payload;
2. Кошелёк свап-пары на который были переведены токены вызывает колбэк свап-пары и передаёт информацию, необходимую для выполнения вывода ликвидности по одному токену;
3. Выполняется операция вывода ликвидности по одному токену из свап-пары;
4. Если операция выполнена успешно, то выведенные токены переводятся на кошелёк, указанный в созданном payload, в ином случае, токены возвращаются на TIP-3 LP кошелёк пользователя;
5. Полученные LP токены сжигаются;



Успешное выполнение операции вывода ликвидности по одному токену

Количество TONов, рекомендуемое для выполнения операций

- Для операции обмена: 0.2 TONa
- Для операции ввода ликвидности по двум токенам:
    - Для первого перевода: 0.2 TONa
    - Для второго перевода: 0.2 TONa, если LP кошелёк существует, 0.4 TONов, если LP кошелька не существует.
- Для операции ввода ликвидности по одному токену: 0.2 TONa, если кошелёк существует, 0.6 TONов, если LP кошелька не существует;
- Для операции вывода ликвидности по двум токенам: 0.8 TONов;
- Для операции вывода ликвидности по одному токену: 0.5 TONов.

Разработанные деботы

Дебот-эксплорер свап-пар

Данный дебот позволяет выполнять следующие операции:

- Получение списка существующих свап-пар;
- Получение информации о выбранной свап-паре;
- Создание новых свап пар;

Деботы для взаимодействия со свап-парой

Для взаимодействия со свап-парой были разработаны ниже представленные деботы:

- Дебот для выполнения операции обмена;
- Дебот для выполнения операции ввода ликвидности по двум токенам;
- Дебот для выполнения операции ввода ликвидности по одному токену;
- Дебот для выполнения операции вывода ликвидности по двум токенам;
- Дебот для выполнения операции вывода ликвидности по одному токену;