svoi
Development
Labs

https://SVOI.dev

Smart contract timer based on FreeTON blockchain Tick-Tock messages

SVOI.dev Labs 2020

General problem description

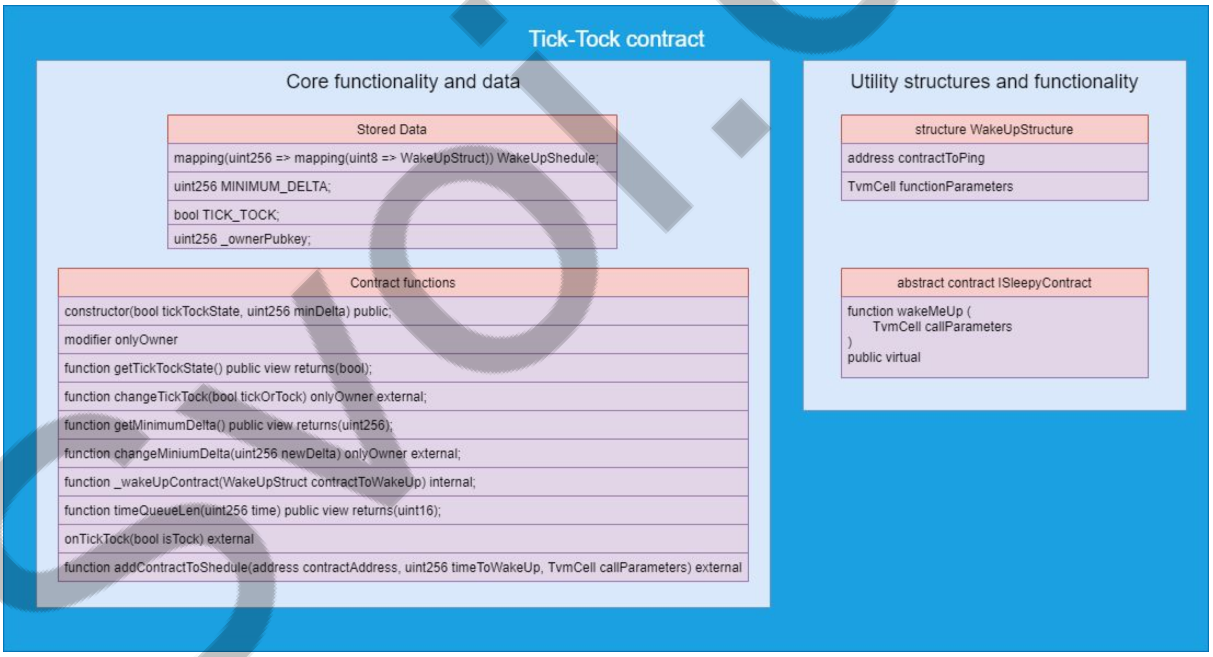
In the blockchain network that allows the implementation of smart contracts, sooner or later smart contracts appear with the need to implement complex logic or work without external influences, such as messages within the TON network. At this point, there is a need for mechanisms that will add external influences for smart contracts. In this case, such a mechanism for providing external influences is the Timer smart contract, which uses special Tick-Tock transactions that exist in the TON network. However, transaction data is only sent to dedicated service smart contracts. In this regard, there is a need to implement a universal timer that will meet most of the needs of programmers when creating products based on smart contracts.

Requirements for interaction with Timer smart-contract

Timer smart contract allows you to wake up a smart contract at a specified time, i.e. call the smart contract method at the specified time.
For this, the smart contract that needs to be woken up must be inherited from the developed abstract smart contract **ISleepyContract**. This abstract smart contract implies the implementation of the **wakeMeUp** method, which allows unifying the interaction of the Timer contract with smart contracts.

Timer smart contract architecture

Since this contract is only the first iteration, it has a fairly simple architecture, which will subsequently be actively developed, without changing the interaction interface to maintain backward compatibility with smart contracts that will start using this timer in the near future.
Below are the main data and functionality, as well as auxiliary data stored and implemented by the smart contract.


Tick-Tock contract
Core functionality and data
Stored Data
Contract functions
Utility structures and functionality
structure WakeUpStructure
abstract contract ISleepyContract

```
mapping(uint256 => mapping(uint8 => WakeUpStruct)) WakeUpShedule;
uint256 MINIMUM_DELTA;
bool TICK_TOCK;
uint256 _ownerPubkey;
```

```
constructor(bool tickTockState, uint256 minDelta) public;
modifier onlyOwner
function getTickTockState() public view returns(bool);
function changeTickTock(bool tickOrTock) onlyOwner external;
function getMinimumDelta() public view returns(uint256);
function changeMiniumDelta(uint256 newDelta) onlyOwner external;
function _wakeUpContract(WakeUpStruct contractToWakeUp) internal;
function timeQueueLen(uint256 time) public view returns(uint16);
onTickTock(bool isTock) external
function addContractToShedule(address contractAddress, uint256 timeToWakeUp, TvmCell callParameters) external
```

```
address contractToPing
TvmCell functionParameters
```

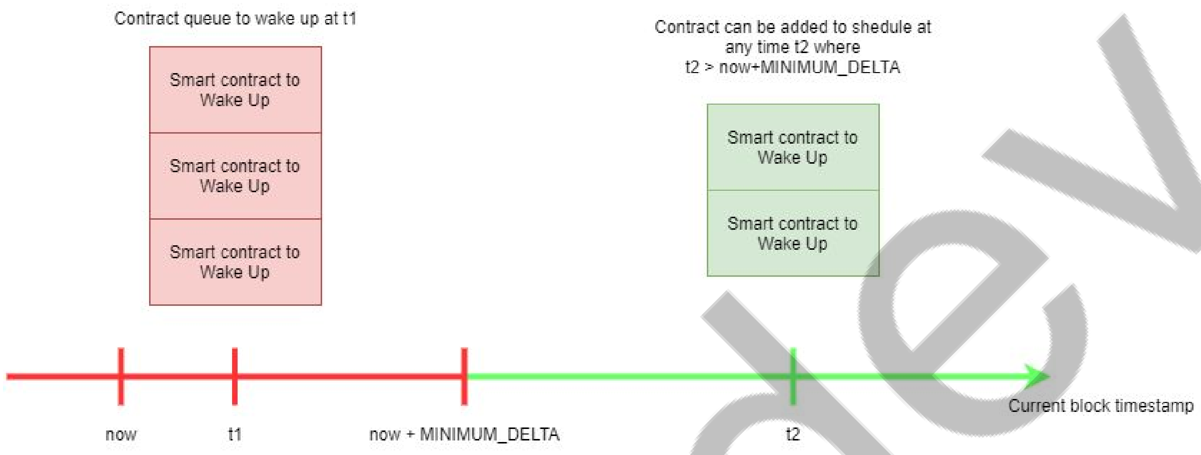
```
function wakeMeUp (
      TvmCell callParameters
)
public virtual
```

Smart contract parameters

As in any smart contract, it is necessary to provide some flexibility to the Timer contract in order to increase the volume of tasks that this smart contract can solve.
This smart contract currently has several settings:

1. The **TICK_TOCK** field, which allows you to select during which transactions the contract will work - TICK or TOCK;

2. **MINIMUM_DELTA** field is a parameter that defines the minimum time interval from the current moment when the add request cannot be fulfilled. I.e. in the time period $[now;\ now + MINIMUM\_DELTA]$ adding a contract for a wake up call with the desired call time t, where $now \leq t \leq MINIMUM\_DELTA$, is not possible. This mechanism is illustrated below.


Contract queue to wake up at t1
Smart contract to Wake Up
Smart contract to Wake Up
Smart contract to Wake Up
Contract can be added to shedule at any time t2 where t2 > now+MINIMUM_DELTA
Smart contract to Wake Up
Smart contract to Wake Up
Current block timestamp
now
t1
now + MINIMUM_DELTA
t2

Data stored in a smart contract

To implement the Timer smart contract, it is necessary to store some information about smart contracts that must be awakened at a given time. This information is stored in the **WakeUpShedule** field of this smart contract.

This field has the following type:

mapping (uint256 => mapping (uint8 => WakeUpStructure))

The choice of this storage option is due to the following factors:

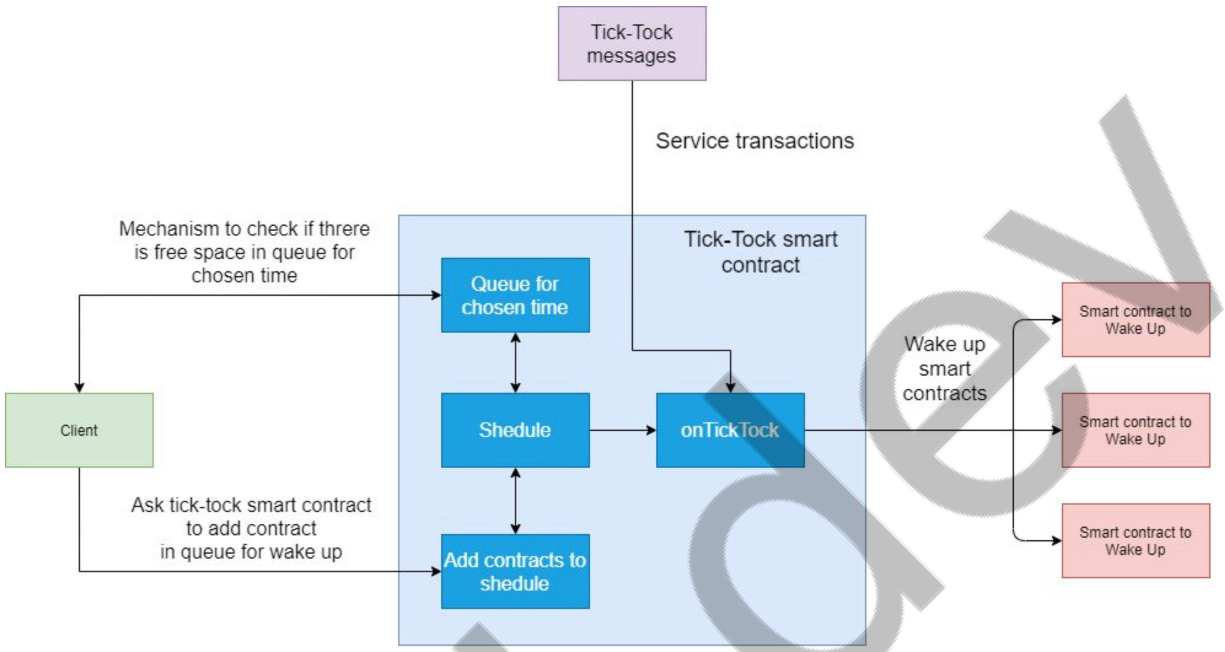
1. The Timestamp of the current block, stored in the now constant, is of type uint256, which makes it easier to interact with time variables;
2. This field is a mapping of the mapping, since at the same time **t** there may be several people who want to call their contract, for example, at the start of a new hour, or, for example, at midnight. The number of smart contracts called at the same time is limited to 256 contracts in order to avoid unnecessary downtime and ensure an acceptable call time not only for the first contract in the queue, but also for all subsequent ones.

Smart contract functionality

This section will be omitted, since it will take up a lot of space and, if necessary, you can familiarize yourself with the source files of the contract, or ask a question on social networks.

Interaction with contract

In this smart contract, interaction with the timer is as follows:


Tick-Tock messages
Service transactions
Mechanism to check if threre is free space in queue for chosen time
Tick-Tock smart contract
Queue for chosen time
Client
Shedule
onTickTock
Wake up smart contracts
Smart contract to Wake Up
Smart contract to Wake Up
Smart contract to Wake Up
Ask tick-tock smart contract to add contract in queue for wake up
Add contracts to shedule

A client wishing to awake a smart contract at a selected time must perform the following actions:

1. Check if there is free spot in the queue for waking up smart contract at the selected time;
2. Request to add smart contract to the wakeup queue.

If the wake-up scheduling request completes without errors, then the spot was reserved successfully. If the request was completed with an error, then the addition did not occur. The error codes will be described below.

This implementation is the first iteration of the Timer smart contract development. In the future, it is planned to modify the smart contract, according to the Roadmap section.

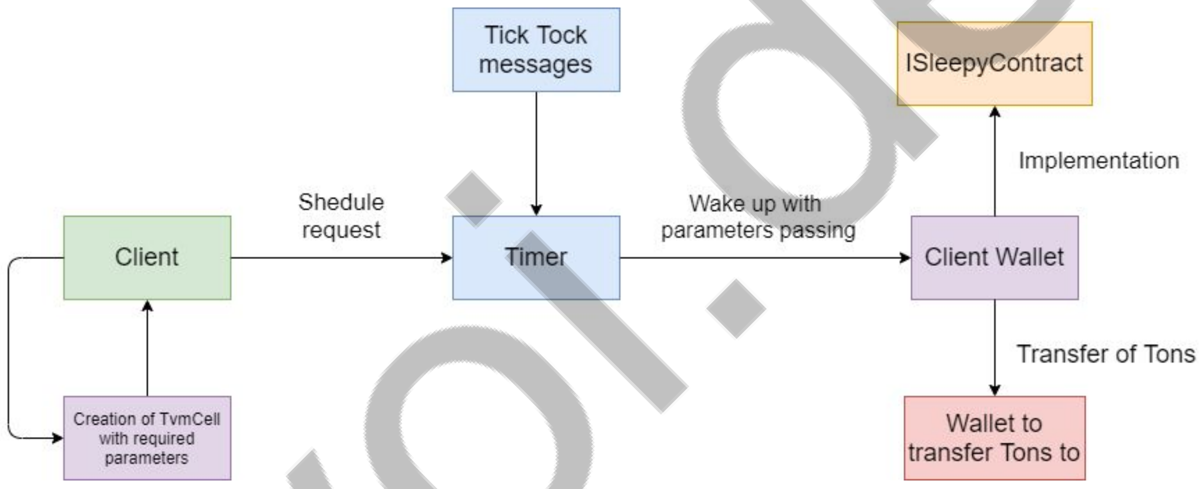
Error codes.

- 101 - incorrect tick/tock message received, if contract is configured to Tick messages, then this error will occur on Tock messages and vice versa;
- 102 - incorrect time to wake up contract, must be greater than $now + MINIMUM\_DELTA$;
- 103 - unauthorized access to some functions, i.e. changing the configuration of smart contract without being the owner;
- 104 - queue full, this error means that there is no free spot left for required time.

Mechanism to notify contracts

As mentioned above, to use this timer, the target smart contract (which needs to be woken up) must be inherited from the **ISleepyContract** interface and implement the **wakeMeUp** function.
This interaction mechanism was chosen due to the fact that in this case, the contract can not only be notified of the occurrence of a certain timestamp, but can also implement complex logic when called, which is achieved by adding a parameter of the **TvmCell** type, which allows you to pack the data required for the subsequent call of the internal functions of the contract, for example, this cell can contain functionId and the parameters of the call of this function.

This fact expands the capabilities of this timer, since it becomes not just a contract notifying other smart contracts about the time stamp, but also allows you to implement deferred calls to complex smart contracts, for example, deferred transfers. A possible implementation of the delayed transfer mechanism is illustrated below.



Timer smart contract accuracy

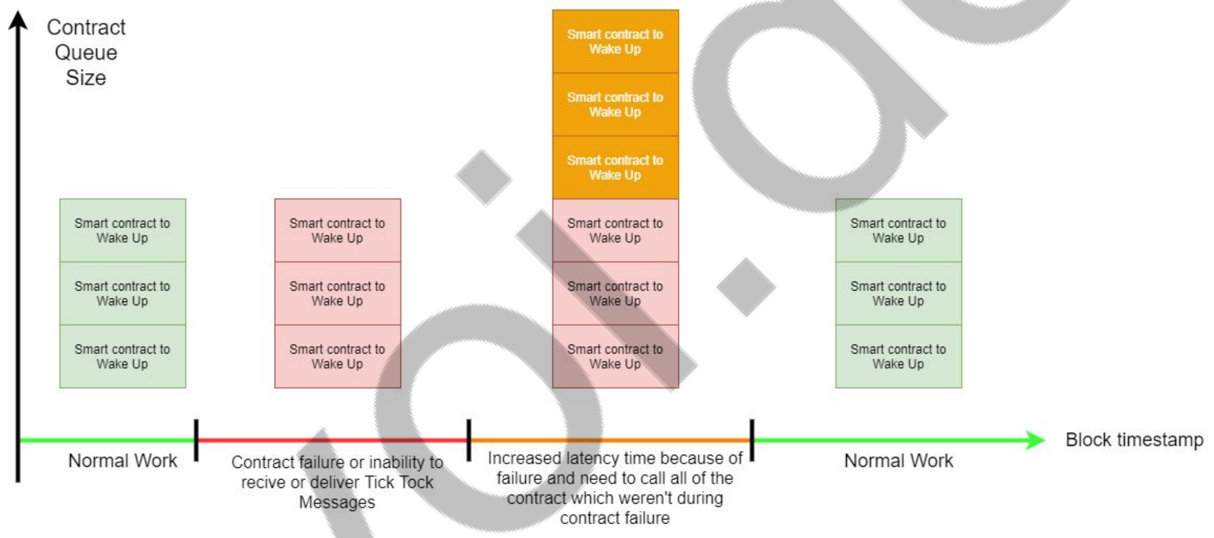
One of the important requirements for a timer smart contract is its accuracy, i.e. what time delay occurs.
The accuracy of this smart contract directly depends on the time required to form the block, since the Tick and Tock messages are formed accordingly at the beginning of the block formation and at the end of the block formation. Also, additional delay arises from the fact that messages are not delivered instantly. The maximum delay time for message delivery (according to the available information) when using hypercubic routing is 4 blocks. Thus, the maximum delay time for a smart contract call is 6 blocks, taking into account the fact that this Timer can be configured for Tock messages that are sent at the end of the block formation, which means that messages sent by the timer will not be included in the current block, but will appear in the next.
At the time of this writing, the block formation time in the main TON network is:

$$t_{new\ block\ creation} = 0.2\ seconds$$

This results in a maximum delay of 1.2 seconds.

Timer smart contract reliability

For a Timer smart contract is not enough to be accurate , if there is no certainty that the contracts will be called upon reaching the specified time stamp, i.e. another requirement is introduced - reliability.
The developed smart contract guarantees that all smart contracts that have reserved a call for a given time will be called. Also, in case of unforeseen delays in the work of a smart contract, i.e. when the T_{wake} call time has been missed, or in case of external circumstances, smart contracts whose call time is less than the block timestamp will be called during the next successful call to the onTickTock function. This point is illustrated below.

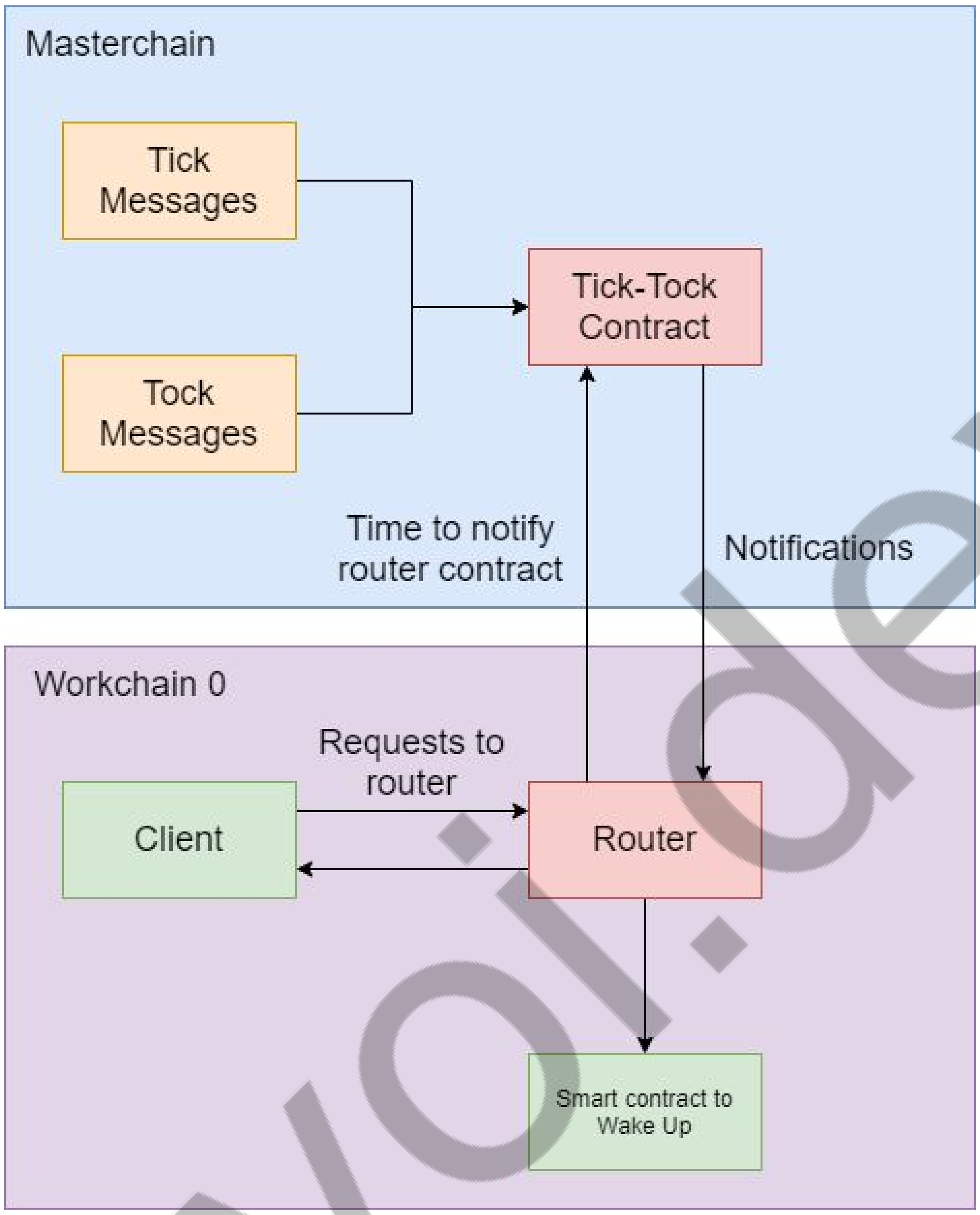

Contract Queue Size
Smart contract to Wake Up
Normal Work
Contract failure or inability to recive or deliver Tick Tock Messages
Increased latency time because of failure and need to call all of the contract which weren't during contract failure
Normal Work
Block timestamp

Roadmap

As mentioned above, this implementation is the first iteration in terms of developing a Timer smart contract.
The development plan for this smart contract will be presented below.

Stage 1

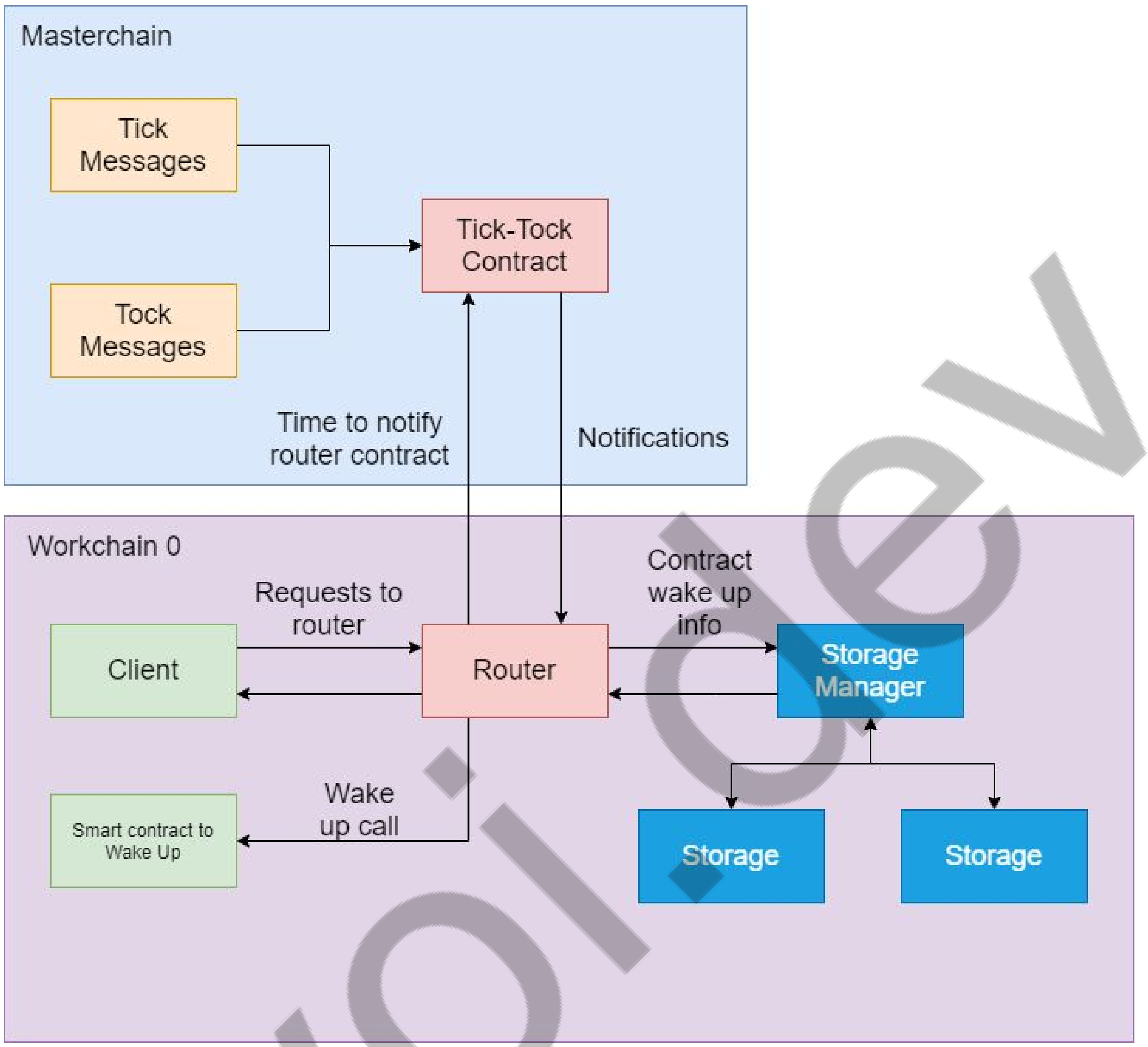
The fee for storing and executing smart contracts in Masterchain is higher than in Workchain 0.
At the first stage, it is planned to divide the smart contract into two parts: a router located in Workchain 0 and a Timer located in Masterchain. All the “heavy” logic will be moved from the smart contract located in the Masterchain to the smart contract located in Workchain 0.



In this case, it is assumed that the batch sending of the time for which the calls to contracts were reserved for the time period $[now; \ now + MINIMUM\_DELTA]$, which will reduce the number of messages sent to the Masterchain. Messages will be sent from **Router** to **TickTockContract** upon request by **TickTockContract** if all calls from the previous batch were made.

Stage 2

In addition to the cost of executing contracts, there is another problem - the limitation of the amount of storage available to smart contracts. In this regard, the amount of information in smart contracts that can be stored in a timer smart contract at one time is limited. Therefore, the next step in the development of a smart contract is to take the stored information out of the contract and create dedicated storage, which will be dynamically created as needed, based on the analysis of the peak and average load on the timer smart contract.



Storage Manager may be a part of Router smart contract or a discrete smart contract.

General Provisions

Also, it is planned to provide regular updates of contracts in accordance with the release of updates and the expansion of the functionality of the FreeTon network.

Project repository

Repository based on GitHub
https://github.com/SVOIcom/TickTockContract