Distributed Validator-Oracle Implementation Description

Pruvendo Team

09/14/21

Executive Summary

This document describes the Pruvendo implementation of the distributed validator-oracle pursuant to the original whitepaper on the NOT as a TON binary system[1] and contest description[2]. As requested, the Pruvendo team have implemented the required NOT a Validator smart contract and NOT an Elector smart contract and the testing system. The solution is implemented in Solidity (the on-chain part) and Python (the off-chain part). Please address all the possible questions to the project lead @SergeyEgorovSPb .

[1] https://firebasestorage.googleapis.com/v0/b/ton-labs.appspot.com/o/documents%2Fapplication%2Fpdf%2F4rdykjz6rxbkk4gmv6k-NOT%20a%20TON%20Binary%20System.pdf

[2] https://forum.freeton.org/t/distributed-validator-oracle-contest/10402

Source

The source code of the solution is available at https://github.com/Pruvendo/not_oracle . The pull request for https://github.com/freeton-org/defi has been also prepared but could not be created due to lack of rights.

1. Description of the solution

The Pruvendo solution consists of the following parts:

- On-chain: the two central smart contracts (NotValidator and NotElector)
- Off-chain: the Python component that queries an exchange for quotations, sends quotations to a NotValidator, and listens to events emitted by the NotValidator

1.1. The on-chain part (the smart contracts)

The on-chain part contains the central smart contracts (as per the contest proposal, NotValidator and NotElector).

1.1.1. NotValidator Implementation

The NotValidator contract (`NotValidator.sol`) registers with a NotElector contract and, provided that the NotValidator is initially accepted by the NotElector, it transmits the TON/USD price feeds to the NotElector.

The lifecycle of the NotElector contract comprises three phases:

1. The election phase. At this stage, persons or entities wishing to act as a validator transfers their stake to NotValidator, calling `transferStake` with a DePool. The DePool calls `onTransfer` with NotValidator, so that NotValidator knows it is the owner of the stake and can participate in the election.
2. The validation phase: once NotValidator has signed up with NotElector, NotElector publishes the information as to which NotValidator applicants were accepted:
    a. If a NotValidator is accepted, it can send quotations to the NotElector, and the latter will accept them.
    b. If NotValidator is not accepted, at the termination of the round, it will be annihilated upon the call of the `onRoundComplete` method. The stake

submitted and the balance will be returned to the owner of the NotValidator contract.

When a NotValidator is accepted by the NotElector, it employs the following methods:

- `setQuotation` sends a hashed and salted quotation received from an external source (exchange) to NotElector and stores the latest quotation
- `requestRevealing` handles NOT an Elector's request to reveal/decrypt the quotation as follows:
    - NotValidator emits a `RevealPlz` event to be received by the off-chain component)
    - Then there are three possible outcomes:
        - NotValidator receives the quotation from the off-chain component and transfers it to NotElector
        - NotValidator finds its latest quotation too old and sends to NotElector the `quotationIsTooOld` response (basically, it says that the NotValidator is not participating in the current round) - in this case, the NotValidator is not penalized
        - NotValidator sends nothing to the NotElector; in this case, NotValidator is penalized, as it is assigned *P=0*.
- `revealQuotation` - called by the off-chain component, whereby the off-chain component transmits to NotValidator the data for decrypting the quotation

3. After validation phase: when the DePool calls `onRoundComplete`, provided that the NotValidator was not found to be malicious, the NotValidator transfers the DePool stake to the original address, transfers the balance to its owner's address, and self-destructs.

1.1.2. NotElector Implementation

The NotElector contract (`NotElector.sol`) receives the quotations from NotValidator contracts, performs the decision-making as to the acceptability of NotValidators and acts accordingly.

1. When an election of NotValidators is active, and a NotValidator contract invokes the `signUp` method, the NotElector accepts the signup request.
2. Once the election time expires (i.e. `endElection` is called, provided that the validation time period has not run out, and `endElection` hasn't been successfully called earlier), NotElector checks whether NotValidator's DePool stake is sufficient, and declines those NotValidators that submitted insufficient stakes. After the election is terminated, no

further NotValidators can apply. All participating NotValidators are notified about the NotElector's decision (via NotValidators' `setIsElected` method call).

3. Accepted NotValidators can then send quotations to NotElector, using the `setQuotation` method.
4. When NotElector transitions into the revealing mode, it requests from NotValidators the quotations they sent, whereby, as far as the revealing mode has not ended:
    a. NotElector requests from NotValidators the decrypted quotations.
    b. NotValidators invoke `revealQuotation` and send decrypted quotations to the NotElector.
        i. If a NotValidator sends a decrypted quote, it is accepted by the NotElector and stored in it
        ii. If a NotValidator declines to participate on the basis, invoking NotElector's `quotationIsTooOld`method, nothing happens, and the rank of such NotValidator does not change.
        iii. If a NotValidator does neither of the above, it is penalized, P=0 being assigned to its expected response.
    c. `calcFinalQuotation` function is used to calculate the current NOT block. This function is invoked either at the completion of the validation time period or upon receiving a quotation from the last remaining validator, whichever happens first.
    d. After this, ranks of NotValidators are re-calculated, and those with insufficient rank are slashed. After `slash` is called on a NotValidator, the NotValidator can no longer interact with the off-chain component.
    e. The final quotation is published.

1.2. The off-chain part

The off-chain part of the Pruvendo solution contains a number of Python scripts. The most significant of these are:

- Deploy_not_validator.py
    - Deploy NotValidator
    - Transfer DePoolStake
    - Register with a NotElector
- Run_not_validation.py
    - Waits until endElection is called by NotElector
    - Retrieves quotations from an external source (exchange)

2. Deployment

2.1. Deployment prerequisites

The deployment prerequisites are as follows:

1. tonos-SE >= 0.28, should you want to launch it locally
2. ton-Solidity-Compiler >= 0.45.0 (used as `$tondev sol compile` in `compile_all.sh`, an alternative path to the compiler's binary can be specified in `compile_all.sh`). The repository also contains all necessary binaries in the `src/artifacts` directory.
3. Python >= 3.9
4. The Python packages mentioned in `requirements.txt`.

2.2. Deployment procedure

After you have fulfilled the deployment requirements, start tonos-SE with the default settings and install the required Python packages:

```
cd src
python3.9 -m venv venv
source venv/bin/activate
python3.9 -m pip install -r requirements.txt

tondev se start
```

The Pruvendo solution package contains two launch scripts:

- `simple_run.sh` launches only one NOT an Elector and one NOT a Validator contract, and one Python script listening to their events at the same time. This is the most lightweight and minimalistic way to launch the whole system. In this case, we receive quotations from cex.io.
- `run_se_tests.sh` launches one NOT an Elector and multiple NOT a Validator contract as separate processes. The quotations are not retrieved from an actual exchange, but from the `test.json` file. `generate_test.py` script generates this data (can be rewritten to check your hypotheses). The events emitted by the NOT an Elector file are logged in result.json. Warning: the test is heavy and places high requirements on the hardware you run it on.