OT-RFC 06

OriginTrail utility market automatic adjustment plugin

Document authors: Miloš Kotlar Phd(c), Branimir Rakić Msc, Djordje Kovačević, Uroš Kukić

Versions:

- 2020-10-16 (v1)
- 2020-10-12 (Draft 4)
- 2020-09-24 (Draft 3)
- 2020-08-18 (Draft 2)
- 2020-07-15 (Draft 1)

Summary

The goal of this Request For Comments (RFC) document is to introduce new tools in the OriginTrail Decentralised Network (ODN) codebase for the decentralized service marketplace, aimed at improving market conditions so as to encourage more data publication offers by the Data Creator (DC) nodes as well as improve the profitability of Data Holder (DH) nodes. This document discusses the issues with the current pricing mechanism which doesn't take the ODN job market fully into account, and introduces a change in the current pricing formula to better reflect the market conditions. We introduce an optional, configurable Automatic adjustment plugin (AAP) node plugin component which will automatically estimate the optimal λ on the network based on the node user configuration.

Problem statement

The current tools to support node service market settings include a pricing formula which doesn't take the current job market into account fully, and only partially involves the value of TRAC token. The current compensation formula is as follows

$$2 \cdot \frac{G_{asprice} \cdot P_{ayoutgas}}{T_{okenvalue}} + \lambda \cdot \sqrt{2 \cdot d_{ays} \cdot M_b}$$

The disadvantage of such a system is that both DH and DC node runners need to manually adjust the price factor λ in order to participate in the market in an optimal way. Additionally due to this, the rapid changes of TRAC prices are detrimental to both the DC and DH nodes, imposing significant effort on node operators to frequently adjust λ to compensate for the volatility.

Rationale

In order for a DH node to maximize returns, we can assume that it is in their interest to receive the highest possible number of service jobs at the highest achievable price. That means that for ***k*** jobs a DH node has received, the total revenue of the node ***R*** equals the sum of all prices P_i of individual jobs chosen for.

$$R = \sum_{i=1}^{k} P_i$$

At any point in time ***t***, a node will only accept job offers for prices $P_i > P_{min}$, where P_{min} is the minimum price a node will accept at time ***t***

Maximizing ***R*** can happen either by

- increasing ***k***, which would be achieved by **decreasing** the P_{min} in order to qualify for more jobs, thus potentially providing scarce services at a suboptimal market price
- **Increasing** P_{min} which will require more TRAC per single job, but will at levels high enough effectively lower the number of jobs received **k**, thus having services underutilized and operating at a suboptimal level

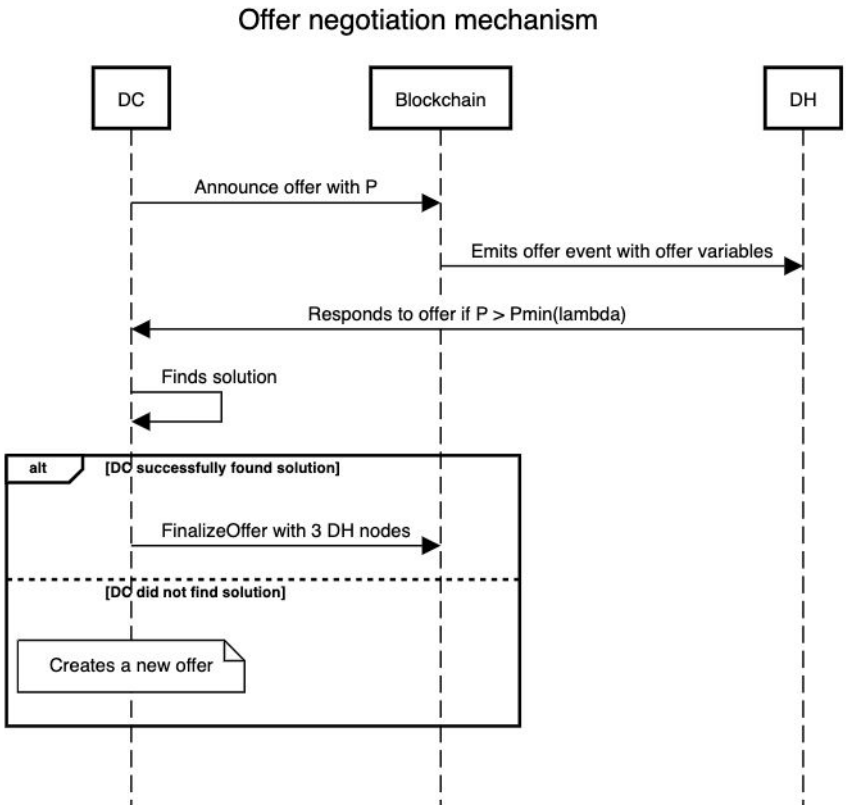
The opposition of the two approaches implies a market "optimal" value for the P_{min} which would maximize the revenue ***R*** at a specific time ***t***.

The formula for ***P*** takes into account several input variables such as the data size, longevity, underlying blockchain transaction cost (currently Ethereum gas price and the constant in gasCost), all of which are dependent on a specific offer and specified at offer creation, **effectively making P_{min} a function of λ from the node operators perspective.**

The current price negotiation protocol maintains that a **DC** node needs to offer a sufficiently high P_i in order to receive enough bids from DH nodes to achieve the necessary degree of decentralization. The collected set of bids from DH nodes needs to be high enough in order for the DC node to be able to "solve" the offer task randomly generated by the ODN smart contracts, which specify the task difficulty according to current network size (number of participants with active staked profiles).
Keeping the P_i too low for DC nodes could result in failed replications as $P_i < P_{min}$ for a sufficient number of DH nodes, effectively having them ignore the offer (as it is too cheap). Therefore the DC node needs to offer a high enough compensation in order for the offer to be accepted by enough DH nodes and it's task be solvable. However an unreasonably high P_{min} on a sufficient number of DH nodes on the network will be a disincentive to DC nodes due to (in DC's view) unreasonably high fees for the service.

Due to the ODN being an open market without centralized control, it is assumed that a fair price for both DH and DC actors will be reached through market forces. The fact that services in the decentralized systems imply a high degree of volatility in short time frames, however, introduces friction in this process.

Therefore, at the current state of the market and protocol, **it is considered valuable to create a more sophisticated set of tools** for "assessing" what the optimal P_{min} at a particular moment in time should be.



Proposed solution

The proposed change is twofold:

1. Implement a change in the current formula to better reflect the market conditions of TRAC. New proposed formula is:

$$P_{min} = 2 \cdot \frac{G_{asprice} \cdot P_{ayoutgas} + \lambda\sqrt{2\ D_{ays}\ M_b}}{Avg(TRAC_{value})}$$

where λ is the "price factor". The entire equation now is inversely proportional to the average TRAC token value over a short and configurable Δt in order to ensure more stability in TRAC price determination and enable the operation of the plugin component

2. Introduce a machine learning plugin component - the AAP plugin - which can intelligently and automatically adjust λ based on available market information in order to minimize the need to constantly adjust the value manually by the node runners.

Automatic adjustment plugin component (AAP)

The AAP component is an optional, configurable **node plugin component which attempts to estimate the optimal λ on the network based on market information.** The goal of this plugin is to optimally set the parameters in order to reach the best compensation for DH nodes and at the same time find the optimal price for DC nodes, helping reach market equilibrium. AAP is able to automatically adjust λ based on historical data, current state and available market information. Each node should have a **model-based plugin component as DH** and a **model-free plugin component as DC.**

Model parameters that are used to assist the node estimation of the current market conditions are using Elo Rating algorithm for an effective learning process. According to the algorithm, model parameters are updated using the formula given below:

$$parameter_i = parameter_{i-1} + \alpha \cdot (actual\ score - expected\ score)$$

Node operator user journey & user experience

The node operator should be allowed to:

1. **Keep the existing price calculation system** (without updating to the new one) to achieve backwards compatibility, but should be aware that they might not be achieving optimal results
2. **Use the new price calculation system** in 2 ways:
   2.1. **Automatic:** easiest, fully autonomous. The system tries to adjust λ as best it can.
   2.2. **Manual:** manually set λ as in previous implementations
3. Select one of the available token value sources implemented in the node client and to be used in the equation. The suggested sources include price Oracles for TRAC/WETH, TRAC/USDC, TRAC/USDT, TRAC/DAI such as Uniswap pairs. For all the sources selected, the average price at a specific moment can be taken.

Finding market equilibrium

The derived formula for market equilibrium of the ODN is:

$$\frac{\sum_{i=0}^{DC_{total}(t)} \left(\frac{\lambda_{DC}[i](t)-\lambda_{DC}[i](t-1)}{\alpha_{DC}[i]} \cdot DH_{total}[i](t)+DH_{answered}[i](t)\right)}{DC_{total}(t)} = \sum_{i=0}^{DH_{total}(t)} \left(\frac{O_{answered}[i](t)}{O_{total}[i](t)} - \frac{\lambda_{DH}[i](t)-\lambda_{DH}[i](t-1)}{\alpha_{DH}[i]}\right)$$

Where:

- DC_{total} - total number of data creators participating in the service market
- DH_{total} - total number of data holders participating in the service market
- $DH_{answered}$ - number of data holders that have answered a specific offer
- $DH_{expected}$ - number of data holders that DC expected to answer on an offer (in relation with risk range)
- α_{DC} - learning rate for data creators λ
- α_{DH} - learning rate for data holders λ
- dc_{change} - degree of change for λ and risk range parameters
- dh_{change} - degree of change for λ and ε parameters
- λ_{DC} - data creator λ factor
- λ_{Dh} - data holder λ factor
- O_{total} - total number of offers
- $O_{answered}$ - number of offers answered
- ε - indicates the deviation from the highest price in the historical time window for data holders

It can be shown that the equilibrium on the ODN decentralized network could also be observed as

$$avg(risk\ range) = avg\ (\varepsilon) \cdot number\ of\ DH\ nodes$$

where the average DC risk range and average DH ε could be defined as the following.

$$\frac{\sum_{i=0}^{totalCreators_t} DH_{expected}[i](t)}{DH_{total}t} = \frac{\sum_{i=0}^{DH_{total}(t)} \varepsilon[i](t\}}{DH_{total}(t)} \cdot DH_{total}(t)$$

The introduced parameters (epsilon and risk range) are derived variables that are used to assist the node estimation of the current market conditions, trend and equilibrium state, effectively enabling the node to perform the learning process.

At any point in time ***t***, any given node operating with the active AAP module will try to correct its λ to find the optimal market price (moving towards the estimated equilibrium). The correction function for the data creator node can be defined as

$$\lambda_{DC}(t+1) = \lambda_{DC}(t) + \alpha_{DC} \cdot dc_{change}(t)$$

where α_{DC} is a learning rate and dc_{change} is

$$dc_{change}(t) = \frac{DH_{expected}(t) - DH_{answered}(t)}{DH_{total}(t)}$$

If the data creator node detects a price change, then $DH_{expected}$ could be defined as

$$DH_{expected}(t+1) = DH_{expected}(t) + \alpha_{DC} \cdot dc_{change}$$

where α_{DC} is learning rate. Correction function for the data holder node can be defined as

$$\lambda_{DH}(t+1) = \lambda_{DH}(t) + \alpha_{DH} \cdot dh_{change}(t)$$

where α_{DH} is the learning rate and dh_{change} is

$$dh_{change}(t) = \frac{O_{answered}(t)}{O_{total}(t)} - \varepsilon(t)$$

If a data holder node detects a price change, then ε could be defined as

$$\varepsilon(t+1) = \varepsilon(t) + \alpha_{DH} \cdot dh_{change}(t)$$

where α_{DH} is a learning rate.

According to the data creator and data holder correlation function definitions, equilibrium on the ODN decentralized network could be defined as the following, where $DH_{expected}$ is

$$DH_{expected}(t) = dc_{change} \cdot DH_{total}(t) + DH_{answered}(t)$$

$$dc_{change} = \frac{\lambda_{DC}(t) - \lambda_{DC}(t-1)}{\alpha_{DC}}$$

$$DH_{expected}(t) = \frac{\lambda_{DC}(t) - \lambda_{DC}(t-1)}{\alpha_{DC}} \cdot DH_{total}(t) + DH_{answered}(t)$$

and ε is

$$\varepsilon(t) = \frac{O_{answered}(t)}{O_{total}(t)} - dh_{change}$$

$$dh_{change} = \frac{\lambda_{DH}(t) - \lambda_{DH}(t-1)}{\alpha_{DH}}$$

$$\varepsilon(t) = \frac{O_{answered}(t)}{O_{total}(t)} - \frac{\lambda_{DH}(t) - \lambda_{DH}(t-1)}{\alpha_{DH}}$$

Finally, equilibrium formula on the ODN decentralized network is

$$\frac{\sum_{i=0}^{DC_{total}(t)} \left(\frac{\lambda_{DC}[i](t) - \lambda_{DC}[i](t-1)}{\alpha_{DC}[i]} \cdot DH_{total}[i](t) + DH_{answered}[i](t) \right)}{DC_{total}(t)} = \frac{\sum_{i=0}^{DH_{total}(t)} \left(\frac{O_{answered}[i](t)}{O_{total}[i](t)} - \frac{\lambda_{DH}[i](t) - \lambda_{DH}[i](t-1)}{\alpha_{DH}[i]} \right)}{DH_{total}(t)} \cdot DH_{total}(t)$$

$$\frac{\sum_{i=0}^{DC_{total}(t)} \left(\frac{\lambda_{DC}[i](t) - \lambda_{DC}[i](t-1)}{\alpha_{DC}[i]} \cdot DH_{total}[i](t) + DH_{answered}[i](t) \right)}{DC_{total}(t)} = \sum_{i=0}^{DH_{total}(t)} \left(\frac{O_{answered}[i](t)}{O_{total}[i](t)} - \frac{\lambda_{DH}[i](t) - \lambda_{DH}[i](t-1)}{\alpha_{DH}[i]} \right)$$

AAP Model-free plugin component for the data creator

The model-free plugin component doesn't require historical data and the goal for the node is to adjust λ in order to successfully deploy an offer on the network. **Risk range** is a parameter which indicates the percentage of the DH nodes required for an offer, meaning how big of a risk the DC is ready to tolerate to have their offers unfulfilled. **Min lambda factor** is a value which is a lower boundary for lambda value. **Alpha** is a learning rate parameter which indicates how fast the model learns. This process can be described as a Markov decision process (MDP) where the model finds equilibrium using the the following correction functions for lambda and risk range:

$$\lambda_{DC}(t+1) = \lambda_{DC}(t) + \alpha_{DC} \cdot dc_{change}(t)$$

$$dc_{change}(t) = \frac{DH_{expected}(t) - DH_{answered}(t)}{DH_{total}(t)}$$

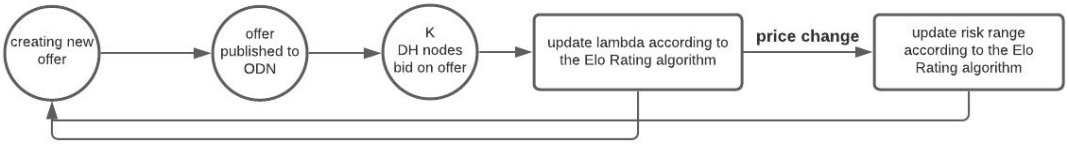
If a data creator node detects a price change, then $DH_{expected}$ could updated as

$$DH_{expected}(t+1) = DH_{expected}(t) + \alpha_{DC} \cdot dc_{change}$$

Markov decision process for DC could be defined as the following:

- **Goal:** Determine the optimal data creator λ parameter for an offer
- **Agent:** DC node making decisions how to change λ parameter in order to successfully publish an offer on the ODN decentralized network
- **Environment:** ODN decentralized network
- **State:** Waiting for an offer, Successfully publish offer on ODN, Offer rejected on ODN
- **Actions:** Update λ parameter for step α, update risk range parameter for step α
- **Reward:** Positive when the DC successfully publishes an offer on ODN and if number of DH nodes are within risk range; Negative when offer has been rejected on ODN or number of DH nodes are is of risk range; Parameter **risk range [0, 1]** presents a safety boundary for the percentage of all DHs bidding.
- **Discount factor:** 0

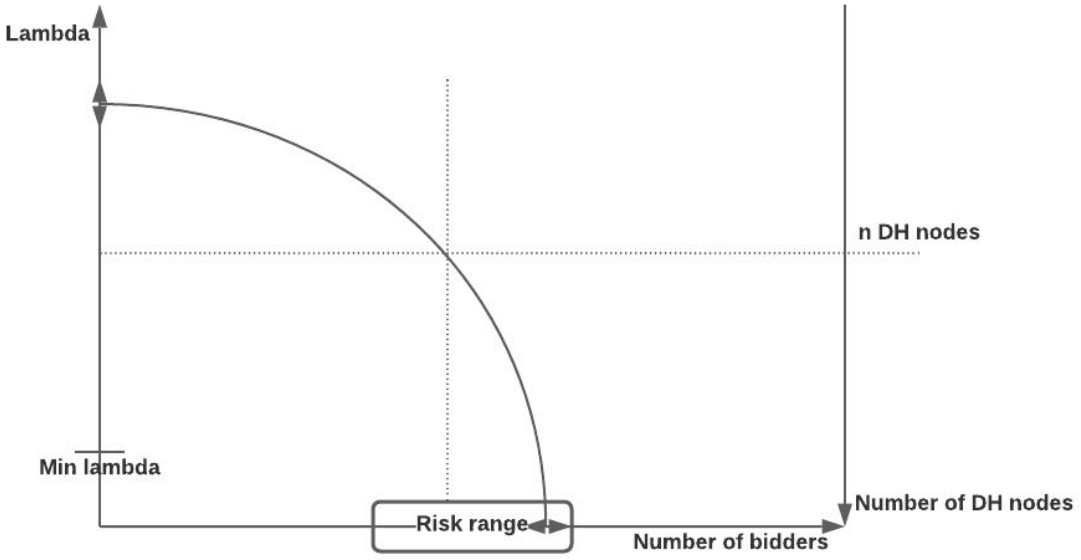
Initial state for DC MDP is creating a new offer. When an offer has been created the DC publishes an offer on the ODN and waits for the outcome. If the offer is successfully published the DC decides to change λ based on offer success and response ratio from DH nodes (percentage of all DHs bidding). If the offer is rejected, DC increases the λ parameter for the alpha step and waits for the next offer to publish. The following diagram shows MDP for DC. Probabilities for states and actions could be calculated using historical data but it doesn't change agents' policy described.



When the DC node detects a change in price on the network by observing the percentage of all DHs bidding, then $DH_{expected}$ is also updated in order to maintain equilibrium on the ODN decentralized network.

The following function illustrates the relation between λ, risk range, and number of DH nodes on the network. With *n* DH nodes on the network, DC nodes are expecting a number of

bidders which is within its risk range. If the number of bidders goes out of the risk range, DC node changes λ and the number of nodes expected within risk range, in order to predict the value of the newly established equilibrium on the network.



The risk range parameter is in relation to the difficulty parameter which determines the probability for finding a solution of an offer task based on the number of bidders for the offer and total number of bidders on the ODN decentralized network.

The difficulty formula

The difficulty of an offer represents the **length of a hexadecimal character string - the task -** that the offer solution needs to contain. The offer solution is a 32 byte value and is generated by successfully finding a tuple of 3 data holder bids hashed in a merkle tree to produce the merkle root which contains the solution. This merkle root solution hash is considered to be sufficiently random, so finding one that contains the required task string is considered to be a series of random number generations. A longer string is harder to generate and requires more input tuples to be used in the attempts to find a tuple that contains the specific value. To understand the current difficulty formula, let's examine the probability that a hexadecimal string of length ***d*** equals a certain combination of hexadecimal characters:

$$P(d) = \frac{1}{16} \cdot \frac{1}{16} \cdots \frac{1}{16} = \left(\frac{1}{16}\right)^d = \frac{1}{16^d}$$

A DC is given a number of bids that generate possible solutions. The probability that there exists a solution that is valid is inversely related to the probability that in all of the possible solutions none are valid. This is why we're going to be dealing with inverses in the following section.

For a hexadecimal string that is 64 characters long (each byte is represented by two hexadecimal characters), the probability of it **not** containing a substring of length ***d*** would be:

$$P'(d) = \left(1 - \frac{1}{16^d}\right)^{65-d}$$

This is the probability that a solution is not contained in the hexadecimal string. Since a possible solution is generated as a merkle tree root of 3 offer bids sent by DH nodes, for ***n*** bids the number of possible solution a DC node can generate is:

$$NumSol(n) = \binom{n}{3} = \frac{n \cdot (n-1) \cdot (n-2)}{6}$$

The probability that none of the possible solutions contain the required substring is then:

$$P'(d, n) = \left(\left(1 - \frac{1}{16^d}\right)^{65-d}\right)^{\frac{n \cdot (n-1) \cdot (n-2)}{6}}$$

The formula for calculating the offer difficulty ***d*** was designed to be in relation to the total number of nodes on the network in such a way that the probability of finding a solution would be extremely low if less than 50% of nodes on the network bid for an offer and to have a probability close to 100% if every node on the network bid for an offer. The currently implemented formula for the offer difficulty ***d*** based on the total number of active nodes on the network ***N*** is:

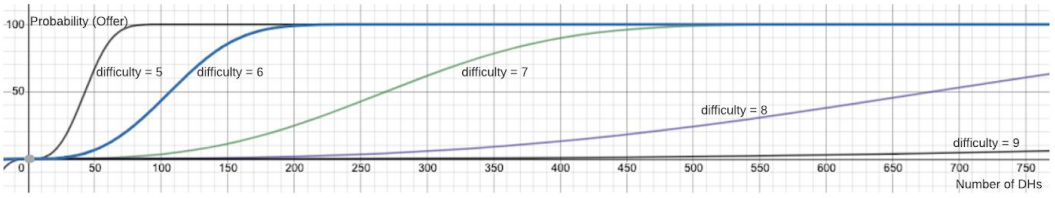
$$d(N) = 4 + \left\lfloor \frac{\lceil log_2(N) \rceil - 4}{1.3219} \right\rfloor$$

For the protocol implementation on Ethereum blockchain, rounding is performed to enable this calculation on the Ethereum Virtual Machine which only deals in unsigned integers The difficulty adjusts on the network based on the network size at any point in time. With the given difficulty formula and the inverse existence probability for a valid solution, the

probability of a DC node finding a valid solution from ***n*** bids on a network that contains ***N*** nodes is:

$$P(N,n) = 1 - \left(\left(1 - \frac{1}{16^{4+\left\lfloor \frac{[log2(N)]-4}{1.3219} \right\rfloor}} \right)^{65-\left(4+\left\lfloor \frac{[log2(N)]-4}{1.3219} \right\rfloor\right)} \right)^{\frac{n\cdot(n-1)\cdot(n-2)}{6}}$$

The following functions illustrate the probability of publishing an offer on the ODN decentralized network for different difficulty values. New difficulty formulas might be proposed in the future.



AAP Model-based plugin component for the data holder

The model-based plugin component should use historical information from the blockchain as training data in order to calculate the optimal value λ, according to the current market conditions. **Refresh rate** is a parameter which indicates how often the model should be updated with new offers from the network. **Epsilon** is a parameter which indicates the deviation from the highest price in the historical time window. Based on this parameter the model can selectively choose which offers to consider as training data (DH chooses best, highest value offers). **Min lambda factor** is a value which is a lower boundary for lambda value. **Alpha** is a learning rate parameter which indicates how fast the model learns.

The training data for the DH model includes blockchain market fees (**gas price, payout gas fee, TRAC in eth), the DC λ, number of days, dataset size, TRAC value (i.e. in USD), offer price only for finalized offers.** The Data sources are ODN smart contracts, OT-Node and price oracles.

Such process can be described as Markov decision process where model finds the equilibrium using the following correction functions for λ and epsilon:

$$\lambda_{DH}(t+1) = \lambda_{DH}(t) + \alpha_{DH} \cdot dh_{change}(t)$$

$$dh_{change}(t) = \frac{O_{answered}(t)}{O_{total}(t)} - \varepsilon(t)$$

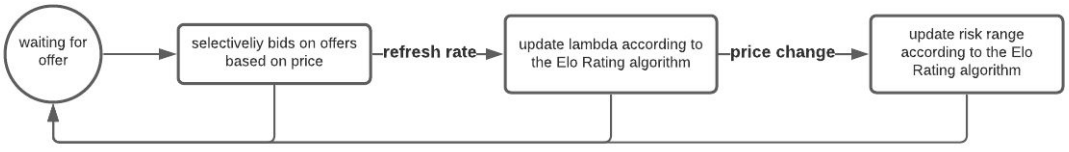
If a data holder node detects a price change, then ε could be defined as

$$\varepsilon(t+1) = \varepsilon(t) + \alpha_{DH} \cdot dh_{change}(t)$$

The Markov decision process proposal for DH could be defined as the following:

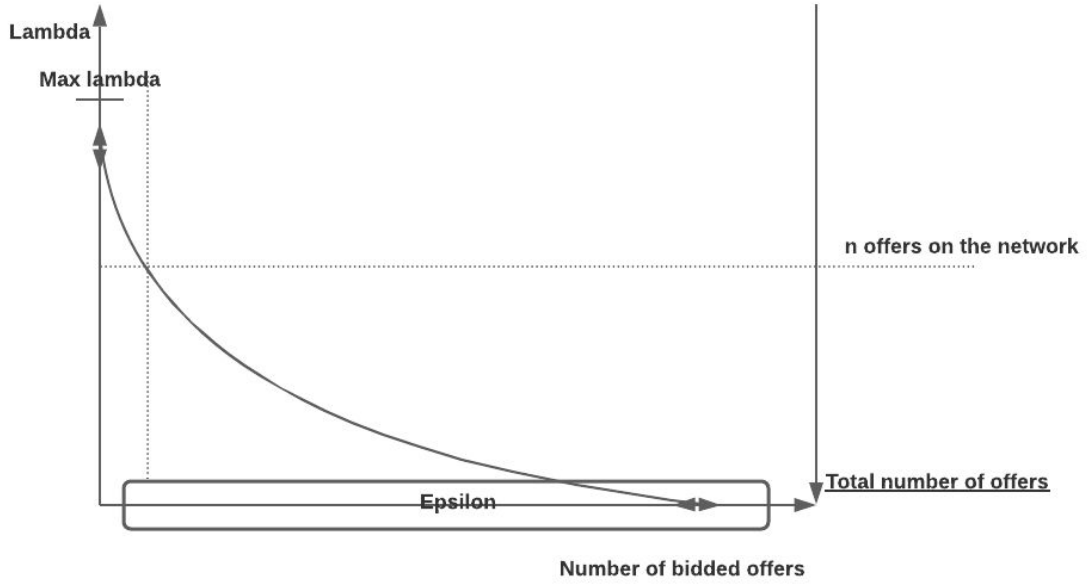
- **Goal:** Determine the optimal data holder λ parameter on the ODN decentralized network
- **Agent:** the DH node makes decisions how to change λ parameter in order to successfully bids on epsilon percentage of offers on the network
- **Environment:** ODN decentralized network
- **State:** Waiting for an offer, selectively bids on best offers, update parameters
- **Actions:** Update λ parameter for step α, update ε parameter for step α
- **Reward:** Positive when the DH successfully bids on ε percentage of offers on the network; Negative when DH bids on more or less than epsilon percentage of offers on the network
- **Discount factor:** 0

The initial state for DH MDP is waiting for an offer. When an offer has been created the DH selectively bids on the offer based on λ value. The DH node decides to change λ value based on ε parameter and the number of bidded offers in the historical time window. If the number of bidded offers isn't within ε parameter the node updates λ parameter for the alpha according to the Elo Rating algorithm. The following diagram shows the MDP for DH. Probabilities for states and actions could be calculated using historical data but it doesn't change agents' policy described.



When DH node detects change in price on the network by percentage of bidded offers on the network, then ε is also updated in order to maintain equilibrium on the ODN decentralized network.

The following function illustrates the relation between λ, epsilon, and number of offers on the network. With *n* offers on the network, DH nodes are expecting a number of bidders which is within ε range. If the number of bidded offers goes out of the ε range, DH node changes λ and ε value within ε range, in order to create a new equilibrium on the network.



Validation simulations

In order to validate the proposed solution the team has performed various simulations, performed in a purpose built AAP simulator (link to the github repository). OriginTrail community members are invited to perform their own simulations and tests and provide the results for public examination and discussion.

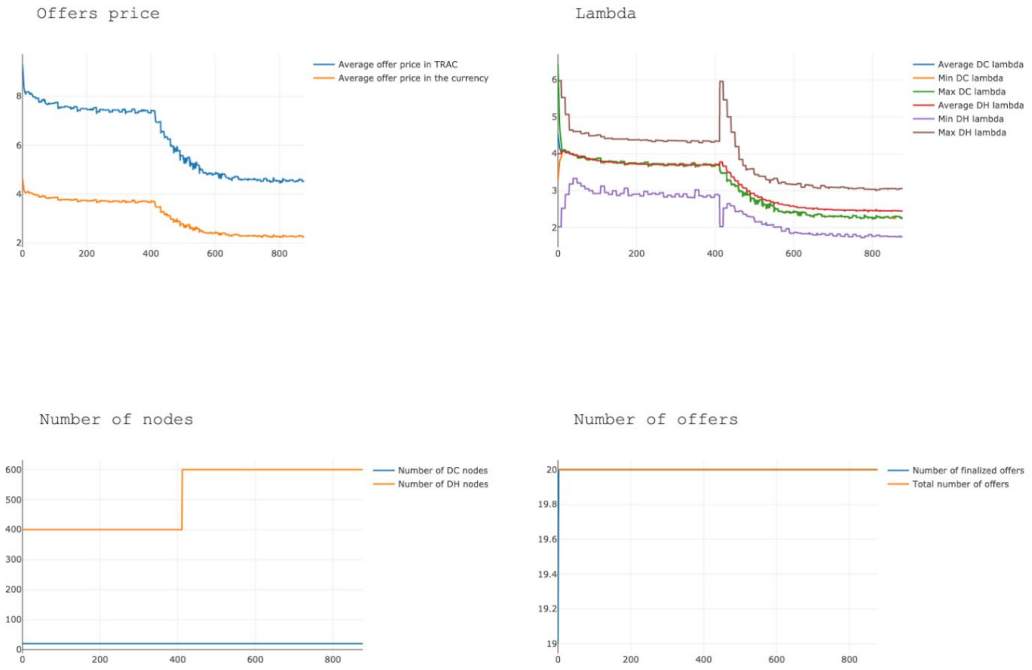
Simulation case 1 - The service market shrinking and equilibrium price going up

When the service market shrinks the equilibrium price can increase, as shown in figures below. The simulation was performed with an initial test set of 20 DC nodes and 400 DH

nodes. When 30% of the DH nodes leave the network equilibrium price goes up and the rest of the network creates a new equilibrium price. **Note that specific values are only indicative and not of significance - the simulation is used only to verify that the system reaches equilibrium with nodes using AAP**.

Simulation case 2 - The service market grows and equilibrium price going down

When the service market grows the equilibrium price can decrease (there are more bidders than previously), as shown in figures below. The simulation was performed with an initial test set of 20 DC nodes and 400 DH nodes. When a set of new 50% of the DH nodes join the network the equilibrium price goes down and the existing nodes with new nodes creates a new equilibrium price. Again, note that specific values for parameters are only indicative and not of significance.



Security considerations

Potential malicious behavior of DC or DH nodes on the ODN may affect the equilibrium price to a certain degree. Normally behaving DC and DH nodes then take actions against malicious nodes to counteract the effect.

Malicious DH nodes

Once the equilibrium price has been achieved on the network, the malicious DH nodes (which would set unreasonable prices, such as very low or very high values) cannot significantly affect the equilibrium price on the network. If a large subset of DH nodes on the network were to become malicious, the rest of the network will start negotiating a new price by excluding the malicious group of DH nodes. After a short period of time a new equilibrium price will be determined. According to the simulations, if a large group of the DH nodes becomes malicious (more than 50% of the network), the equilibrium price will be slightly increased. In such a case, normally behaving DH nodes will overtake offers from malicious DH nodes, while malicious nodes will not receive any offers.

Malicious DC nodes

Similarly to DH nodes, the malicious DC nodes can't significantly affect the equilibrium price on the network. However, if a large subset of DC nodes on the network become malicious (e.g. lower prices dramatically to "convince" the DH nodes that they should lower their prices too), then the rest of the network will start negotiating a new price by excluding the malicious group of DC nodes. After a short period of time a new equilibrium price will be determined. According to the simulations, if a large group of the DC nodes becomes malicious (more than 50% of the network) and publishes offers with low price, the equilibrium price will only be slightly decreased. In such a case, malicious DC nodes will unsuccessfully publish offers on the network which can present a significant cost for them depending on the underlying blockchain implementation.

Implementation

Implementation is expected to commence after careful system inspection, validation and RFC discussion.

Backwards compatibility

As price calculation is not integral to the protocol, there should be no backwards compatibility issues if proper node configuration is in place. The core development team will be issuing detailed upgrade instructions, should any be performed prior to deciding to use AAP.

Important considerations & known issues

A sufficiently long offer will potentially have a disparity between the withdrawal price component (due to gas price volatility) at offer start and end, which is to be addressed in a future RFC.

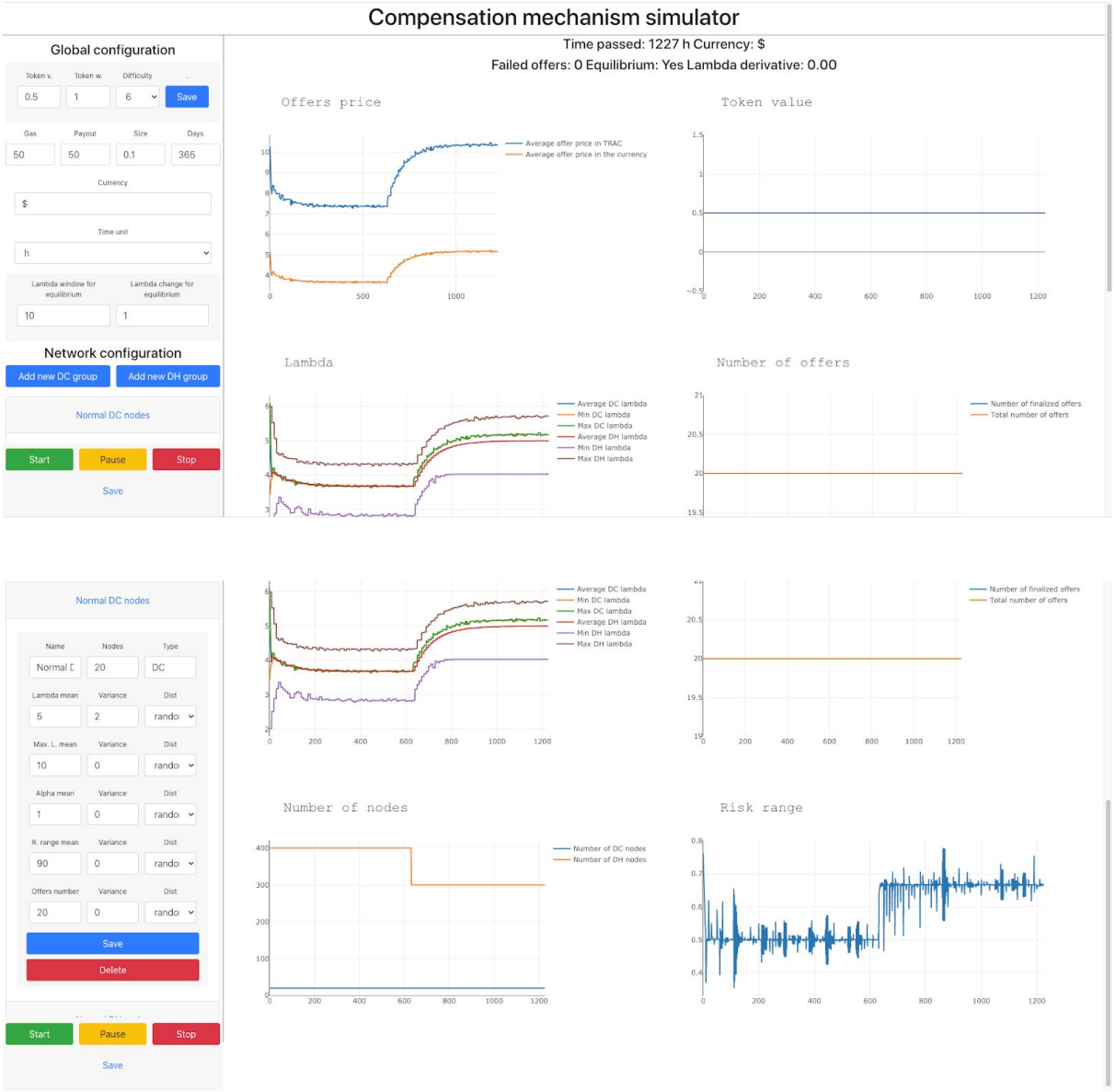
AAP Market Simulator

The goal of the AAP simulator created by the core development team is to enable easy performing of market condition simulations to determine any flaws in the mechanism,

validate the proposed system and to show malicious simulation results and verdict on the safety of the mechanism.

Link to the public github repository: https://github.com/OriginTrail/lambda-simulator-private

The following figures present results of a simulation where the user is able to modify the model parameters for a group of nodes.



We invite community members to try their own simulations of the proposed system and send snapshots to the core developers on the official Github simulator repo as Github issues. In order to create a snapshot of a simulation, click on the “Save” link at the bottom left of the simulator. Graph images are saved in the downloads directory while simulation

snapshots are located in snapshots directory, which is located in the root directory of the project.

References

An example of Markov decision process:

Zheng, Jianjun, and Akbar Siami Namin. "A markov decision process to determine optimal policies in moving target." *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2018.

An example of Elo Rating algorithm:

Mangaroska, Katerina, Boban Vesin, and Michail Giannakos. "Elo-rating method: towards adaptive assessment in e-learning." *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. Vol. 2161. IEEE, 2019.

OriginTrail Official documentation: http://docs.origintrail.io/en/latest/

AAP Simulator github repository: https://github.com/OriginTrail/AAP-market-simulator

OriginTrail whitepaper: https://origintrail.io/storage/documents/OriginTrail-White-Paper.pdf