The on-chain claiming mechanism economics

Piotr Kosiński*
ChronoLogic

May 25, 2018
v1.0

Abstract

Running a service like a TimeNode can be profitable for the operators, however, given the many variables that influence its payout, it's not trivial to estimate the expected profits. The proposed model and simulation code has been created to address that problem.

1 Introduction

This article describes the economic model for on-chain claiming mechanism used by the Ethereum Alarm Clock protocol. This mechanism has been implemented in order to improve the economics of running the **TimeNode**[1].

Given that scheduled transactions are expected to be executed at the exact time and the network of n competing nodes exists, we expect to face the "swarming" problem which can be described as: *uncoordinated attempts of execution by n nodes at the same time*. This problem may result in unnecessary costs[2] for **TimeNode** making the operations potentially not profitable as only 1-of-n is going to earn the TimeBounty for the execution, other **TimeNodes** trying to execute at the same block will pay the failing transaction cost.

By introducing payment modifier P_{mod} **TimeNode** operators are able to pick theirs profitability point, this effectively allows to mitigate the "swarming" problem as we expect **TimeNodes** to try to claim on different blocks/moment of time.

2 Claiming mechanism

Claiming mechanism can be described as follows: For any transaction Tx that has been deployed to the network and is expected to be executed by 1 of n nodes

*Thanks to Vijay Kandy, Piper Merriam, Logan Saether for the contribution.

[1]TimeNode is an off-chain executing agent that scans the blockchain for new scheduled transactions and attempts the execution

[2]due to the fact that all transactions on the Ethereum network costs gas, successful and failed

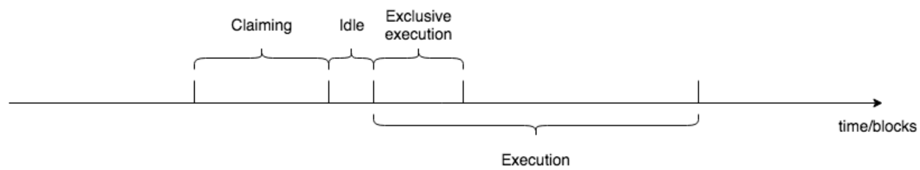

Figure 1: Scheduled transaction life cycle

in the network. The process of execution is divided into two steps: **claiming** and **execution**. Claiming is a process of reserving the transaction for further execution.

2.1 Claiming

- can happen before execution
- every node has the same chance to successfully claim Tx
- claiming requires a *Deposit* to be locked by claimant
- *Deposit* is lost by claimant when execution won't happen within *exclusive execution window*
- every node can fail on claiming when Tx was already claimed
- claiming requires sending transaction that has cost described as C_c when success and C_f when failure
- payment modifier $P_{mod}(t) = \begin{cases} 0 & \text{at beginning of claiming window} \\ 1 & \text{at end of claiming window} \end{cases}$
- claiming is optional

2.2 Execution

- successful execution has reward described as $TimeBounty$
- execution cost is reimbursed by the scheduler when successful
- execution cost has cost described as C_e when not successful
- *Deposit* locked by the claimant can be acquired by a node when the claimant failed to execute

2.3 Expected payout definition

Let's define the expected payout for node as

$$P = P_s + P_f + P_{nf}$$

where

P_s is expected payout after successful claiming and execution

P_f is expected payout after successful claiming and missed execution

P_{nf} is expected payout after other node losing the deposit

2.3.1 Network with $n = 1$ nodes

For network of nodes with $n = 1$ we can define expected payouts as:

$$P_s(P_{mod}) = P_{mod} \times TimeBounty - C_C$$
$$P_f = -C_C - Deposit$$
$$P_{nf} = TimeBounty - C_C$$

2.3.2 Network with $n > 1$ nodes

For that case the expected reward will be $\frac{TimeBounty}{n}$ assuming that probability of successful claiming is equal for all nodes. Also in case of failing transaction node will pay the C_{Tx}

$$P_s(P_{mod}) = \frac{P_{mod} \times TimeBounty - C_C}{n} - (n-1) \times C_{Tx}$$
$$P_f = -C_C - Deposit$$
$$P_{nf} = \frac{TimeBounty - C_C}{n} - (n-1) \times C_{Tx}$$

In order to improve the cost of failing transactions let's introduce a mechanism X that prevents sending the transaction that will fail, the accuracy of mechanism X is defined as $A_X \in [0; 1]$

$$P_s(A_X, P_{mod}) = \frac{P_{mod} \times TimeBounty - C_C}{n} - (1 - A_X) \times (n-1) \times C_{Tx}$$
$$P_f = -C_C - Deposit$$
$$P_{nf}(A_X) = \frac{TimeBounty - C_C}{n} - (1 - A_X) \times (n-1) \times C_{Tx}$$

The last part is to introduce $P_{ld} \in [0; 1]$ as probability of TimeNode loosing the *Deposit*

$$P(A_X, P_{ld}, P_{mod}) = P_s(A_X, P_{mod}) \times (1 - P_{ld}) + (P_f(A_X) + P_{nf}(A_X)) \times P_{ld}$$

3 Simulation

We are going to simulate few cases using equation formulated in section 2.3.2. A_X, P_{ld}, P_{mod} are the variables that depends on reliability and running costs by TimeNode owners. All calculated results are going to be represented as *gas*

Let' now define the profitability threshold, we assume monthly running cost for the TimeNode is 7USD (this is based on current rates on Heroku cloud). Translating 7USD to *gas* we get:

$$ETH/USD = 700USD$$
$$GasPrice = 10Gwei$$
$$7USD \equiv 1000000gas$$

This shows us that running costs are covered after acquiring 1000000*gas*. Now let's take a look at how this translates to the amount of executed transactions. In order to calculate $P(A_X, P_{ld}, P_{mod})$ we use the script listed below. More over we are using these values describing TimeNode operations and network conditions.

$$n \approx 8$$
$$TimeBounty \approx 300000gas$$
$$Deposit \approx 600000gas$$
$$A_X \approx 0.95(95\%)$$
$$P_{ld} \approx 0.1(1\%)$$
$$C_C = 90000gas$$
$$C_{Tx} = 25000gas$$
$$target = 1000000gas\ or\ 7USD$$

Results using these parameters are:

```
   p_mod      res num_of_tx
16  0.75 1164.673       858
17  0.80 2442.108       409
18  0.85 3957.999       252
19  0.90 5381.834       185
20  0.95 6967.551       143
21  1.00 8388.889       119
```

Results data frame contains 3 columns:

- **p_mod** - payment modifier (P_{mod})
- **res** - expected amount of gas earned by TimeNode per transaction

- **num_of_tx** - number of transactions to be executed in order to cover running costs

The results achieved by running this simulation should be treated informational rather than something taken for granted. The expected payout depends on all of the variables described above, for a simulation purpose we picked values using our intuition.

The variable controlled by the TimeNode operator is P_{mod} which is the major component. Low enough P_{mod} allows TimeNode to claim transaction before others, still, in order to be profitable using low P_{mod}, the TimeNode running cost has to be low.

The TimeNode market has many characteristics of the perfect competition market[3]: there is perfect information, no barriers to entry, they deliver the same service, TimeNode are the price takers. Based on that, long-term we may get in the situation where marginal cost is equal average cost.

A Simulation source code

Listing 1: Siml.R

```
set.seed(123)

tx_cost <- function(A_x, n, C_tx) {
  (1 - A_x) * (n - 1) * C_tx
}

bounty <- function(timebounty, C_c, n) {
  (timebounty - C_c) / n
}

P_s <- function(P_mod, timebounty, C_c, n, A_x, C_tx) {
  P_mod * bounty(timebounty, C_c, n) - tx_cost(A_x, n, C_tx)
}

P_f <- function(C_c, deposit) {
  -C_c - deposit
}

P_nf <- function(timebounty, C_c, n, A_x, C_tx) {
  bounty(timebounty, C_c, n) - tx_cost(A_x, n, C_tx)
}

outcome <- function(p_ld,
                    A_x,
```

[3] http://www.economicsonline.co.uk/Business_economics/Perfect_competition.html

```
                    n,
                    C_tx,
                    P_mod,
                    timebounty,
                    C_c,
                    deposit) {
  P_s(P_mod, timebounty, C_c, n, A_x, C_tx) * (1 - p_ld) + (P_f(C_c, deposit) +
}

#Normal distrubutions
timebounties = rnorm(1:1000, mean = 300000, sd = 50000)
deposits = rnorm(1:1000, mean = 600000, sd = 50000)

network = rnorm(1:1000, mean = 8, sd = 2)
network = as.integer(network)
network[network <= 0] = 1

accs = rnorm(1:1000, mean = 0.95, sd = 0.05)
accs[accs > 1] = 1

p_lds = rnorm(1:1000, mean = 0.02, sd = 0.01)
p_lds[p_lds < 0] = 0

c_c = 90000 #claiming cost
c_tx = 25000 #failing tx cost

p_mods = seq(0, 1, 0.05) #payment modifiers

simulate <- function() {
  result = c()

  p_mods = seq(0, 1, 0.05)
  for (p_mod in p_mods) {
    out = c()
    for (i in 1:10000) {
      A_x = sample(accs, 1)
      p_ld = sample(p_lds, 1)
      n = sample(network, 1)
      deposit = sample(deposits, 1)
      timebounty = sample(timebounties, 1)

      out[i] = outcome(p_ld,
                       A_x,
                       n,
                       c_tx,
                       p_mod,
```

```
                        timebounty,
                        c_c,
                        deposit)
    }
    result <- c(result, mean(out))
  }

  result
}

results = simulate()
target_gas = 1000000

x = data.frame(p_mod = p_mods, res = results, tx = as.integer(target_gas / resul
subset(x, res > 0)
```