Ardana Rollups On-Chain Transaction Pricing

Ardana Labs

December 24, 2021

A key bottleneck in Ardana rollups is the process of moving funds from the chain to the rollup and back again. To add funds to the rollup, a user sends the funds to the rollup contract address, where they will be picked up and incorporated into the rollup by an on-chain "input" transaction posted by the prover. To remove funds from the rollup, a user posts an on-rollup removal transaction, and then the prover creates an on-chain "output" transaction which processes several removal transactions, sending the funds to their owners' wallet adresses. These input and output transactions are expected to be a bottleneck because of the limited throughput of the chain.

We wish to ensure the availability of rollup services, to the greatest extent possible, and this includes the services of adding and removing funds. However, we are strictly limited in our ability to increase throughput for adding and removing funds. Therefore, in order to ensure the availability of these services, we will use a pricing mechanism for these services which balances supply and demand. If demand for the services exceeds the supply, then we will raise prices so that demand does not exceed supply. If supply for the services exceeds the demand by a lot, then we will lower prices so that supply does not exceed demand so much, unless the price cannot be lowered any further.

The minimum cost of adding and removing funds is the network fees charged by the blockchain, plus the cost of creating the on-rollup removal transaction (in the case of removals). These costs are out of scope of the current proposal. These costs do not serve to balance supply and demand, but rather just to cover the costs of operating the network. The current proposal describes additional charges which are not to cover operation costs but rather to balance supply with demand. These additional charges can be zero, and there is no limit to how high they can be. We will refer to these

additional charges as the "availability surcharge."

We will provide on-chain rollup services at four different price levels, which we will call free (0), low (lo), medium (med), and high (hi). These prices are not required to be different; they can each be zero or any non-negative real number. Nor do we require one to be greater than another; it is for example possible to have $hi < med < lo$.

Low, medium, and high price levels are supposed to track the market price of transactions with a certain *target latency*. Latency means the amount of time between a user requesting a transaction and that transaction settling. The true value of the latency is unique for each transaction, and not possible to measure exactly, and thus not a known quantity but only an estimated quantity.

Here is an example of what the target latency values could be:

Price level	**Target latency**
High	2 blocks
Medium	10 blocks
Low	100 blocks

The target latency is not a guaranteed transaction latency maximum or service level agreement (SLA). Instead, the target latency is the *target mean transaction latency*. This means that the pricing algorithm attempts to bring the mean of the transaction latency for transactions at this price towards the target mean transaction latency. There is no guarantee or SLA for either the mean transaction latency or the maximum transaction latency at any price level. The latency depends on factors outside of our control, such as overall demand for chain throughput. Therefore, this pricing algorithm proposal cannot offer any guarantee or SLA about the latency with which on-chain rollup contract transactions are settled. What this proposal offers is a good faith effort to cause the target mean latency to be approached for each price level at all times.

It is possible to cause the target mean latency to continuously be approached by considering latency as a function of supply and demand. When the rate of demand is faster than the rate of supply, this leads to longer wait times. In other words, when demand exceeds supply, the queue of requests waiting to be settled becomes longer, and thus, latency increases until the increasing latency slows the rate of demand.

That process of latency increasing when the rate of demand exceeds the rate of supply is what the pricing algorithm is intended to stop. This is based

on an assumption that we can control demand by adjusting price. As price increases, the rate of demand falls.

We are considering a pricing problem where the rate of supply is variable and outside our control, the rate of demand is variable and outside our control, and we control the price. By controlling the price, we indirectly control the rate of demand. The rate of supply is assumed to be fully outside our control for the purposes of this algorithm, being dependent on the condition of the blockchain network.

We could just as accurately call this pricing algorithm the latency-stabilizing supply-demand equalizing algorithm. It aims to stabilize the latency by equalizing supply and demand by controlling the price. We do so at three different price levels which correspond to three different latency levels.

This proposal also offers a free price tier, where the availability surcharge is guaranteed to be zero, but there is no target mean transaction latency. Without a price, we have no means of continuously approaching a target mean transaction latency, because we have no means of affecting demand other than latency itself.

For the three non-free price tiers, the prices are computed independently of each other, each by the same algorithm.

1 Computing the availability surcharge for one price level

The availability surcharge will be computed off-chain by the prover. The prover will periodically write the value of the availability surcharge (denominated in ADA) to the verifier contract state UTXO via a transaction which must be signed by the prover.

The initial value of the availability surcharge will be zero. Changes to the availability surcharge will be based on between the mean estimated latency for transactions occurring in the last t seconds, over the target mean transaction latency. Call this ratio r_t. By definition, $r_t = m_t/M$, where m_t is the mean of the estimate latency for transactions occurring in the last t seconds and M is the target mean transaction latency.

The prover will continuously compute r_{5m} and r_{1h}, i.e. r for 5 minutes and 1 hour ($m = 60, h = 60 \times 60$). If $r_{5m} > 1.2$, then the prover will raise the availability surcharge. If $r_{5m} < 1$ and $r_{1h} < 0.75$, then the prover will

lower the availability surcharge.

When the prover raises the availability surcharge, it will do so by the following algorithm. If the availability surcharge is 0, then the prover will raise it to 1 ADA. If the availability surchage is $x > 0$, then the prover will raise it to $1.5x$.

When the prover lowers the availiabilty surcharge, it will do so by the following algorithm. If the availability surcharge is less than 0.01 ADA, then the prover will lower it to 0. Otherwise, if the availability surcharge is x, then the prover will lower it to $0.75x$.

The prover will not change the availability surcharge more than once every five minutes.

This algorithm is designed to respond quickly to increases in demand, and respond more slowly to decreases in demand. This proposal assumes that surges in demand may happen quickly but are likely to persist for a while after they happen. It also assumes that it is better to overcharge on the availability surcharge than to have demand exceeding supply for a sustained period (which may lead to significant delays in processing input and output transactions).

There are separate availability surcharges for adding and removing funds. Supply and demand for these two transaction types are assumed to be different. The availability surcharge for each of the two on-chain transaction types (addition and removal) is computed by the same algorithm, but these two surcharges float independently of each other.

2 Generalizing to multiple price levels

For each price level, we separately track the mean estimated latency of transactions. Thus r_t can be computed separately for each price level; we have r_t^{lo}, r_t^{med}, and r_t^{hi}.

If we had only one price level, then we would devote all throughput of the rollup contract to that price level, and therefore we could compute the number of requests processed for that price level as the number of requests processed in the time period. However, with multiple price levels, we need to further specify how we go about deciding how many requests of each price level go into each on-chain transaction.

Essentially, we are working with a fixed throughput capacity. In fact, it will vary over time, due to variations in network congestion. However, it

remains that the number of requests we were able to process in the last t seconds is likely to be a good estimate of the number of requests will be able to process in the next t seconds. So, when we are trying to pack requests into a block, we are working with an amount of throughput, or in other words a rate of supply, which we can assume we can usually predict fairly accurately.

Let n be the number of requests which we expect we can settle in the next t seconds. Let $n_0, n_{lo}, n_{med}, n_{hi}$ be the number of transactions which we intend to settle in the next t seconds. We require:

$$n = \sum_{i \in \{0, lo, med, hi\}} n_i. \tag{1}$$

How do we achieve this? The simple recommendation here is to require:

$$\text{for all } i, j \in \{0, lo, med, hi\}, n_i \in \{n_j + k \mid k \in \{0, \pm 1\}\}. \tag{2}$$

In other words, we require that the allocation of the transaction throughput expected to be available is as close to equal as possible. This would work about as well for our purposes as any other fixed allocation. Compared to using some sort of variable allocation scheme, using a fixed allocation simplifies the analysis of the algorithm by reducing the number of variables. This allocation seems to Ardana to be fair and equitable, ensuring that services will be available at all price levels if they are available at any price.