Pricing Non-fungible Resources

Toward Multi-dimensional Fee Markets

Theo Diamandis

Bain Capital Crypto & MIT

Joint work with Alex Evans, Tarun Chitra, Guillermo Angeris

Fee markets with a joint unit of account are inefficient.

Our work: framework to optimally set multi-dimensional fees.

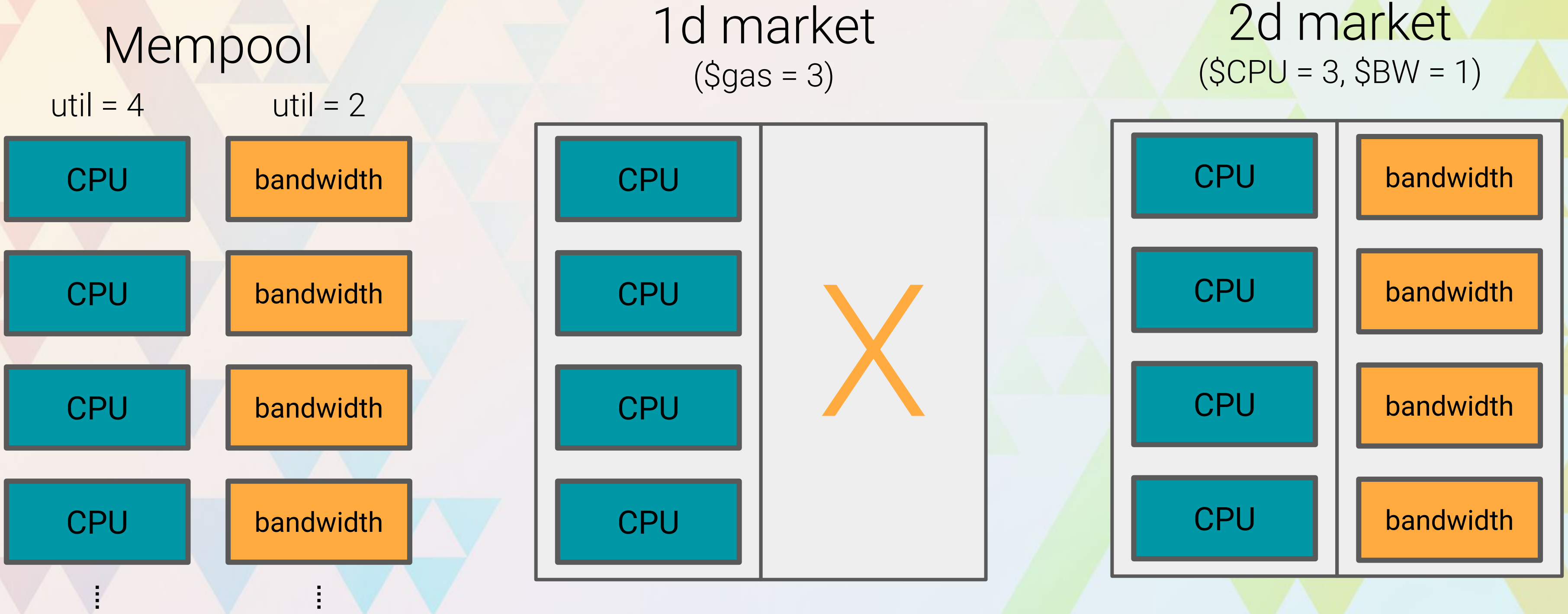
Section 1

Why are transactions so expensive?

Fixed unit of account leads to DoS attacks

- All opcodes have fixed prices relative to each other
- Potential mismatch between relative prices & resource usage leads to ***resource exhaustion attacks*** (DoS attacks)
    - EXTCODESIZE attack in 2016 exploited disk read mispricing
    - Opcode costs manually adjusted (EIP-150)

Fixed unit of account limits throughput



Orthogonal resources should be priced separately.

We need a mechanism for resource **price discovery!**

But what is a resource?

- Anything that can be **metered**
  - Blobs (EIP-2242 & EIP-4844)
  - Compute, memory, storage
  - Opcodes
  - Sequences of opcodes
  - Compute on a specific core
  - ...

Let's formalize this

- A transaction **j** consumes a vector of resources $\mathbf{a}_j \in \mathbb{R}^m$
  - $(\mathbf{a}_j)_i$ = amt of resource **i** consumed by tx **j**

Let's formalize this

- A transaction **j** consumes a vector of resources $\mathbf{a}_j \in \mathbb{R}^m$
  - $(\mathbf{a}_j)_i$ = amt of resource **i** consumed by tx **j**
- $\mathbf{x} \in \{0, 1\}^n$ records which of **n** possible tx's included in block
  - $\mathbf{x}_j = \mathbf{1}$ if tx **j** included, **0** otherwise

Let's formalize this

- A transaction $\mathbf{j}$ consumes a vector of resources $\mathbf{a}_j \in \mathbb{R}^m$
  - $(\mathbf{a}_j)_i$ = amt of resource $\mathbf{i}$ consumed by tx $\mathbf{j}$
- $\mathbf{x} \in \{0, 1\}^n$ records which of $\mathbf{n}$ possible tx's included in block
  - $\mathbf{x}_j = \mathbf{1}$ if tx $\mathbf{j}$ included, $\mathbf{0}$ otherwise
- The quantity of resources consumed by this block is then

$$y = \sum_{j=1}^{n} x_j a_j = Ax$$

We constrain & charge for each resource

- Define a target resource **$b^\star$**
    - Deviation from the target is **$Ax - b^\star$**
    - In Ethereum: **$b^\star$ = 15M gas**

We constrain & charge for each resource

- Define a resource target $\mathbf{b}^\star$
  - Deviation from the target is $\mathbf{Ax - b}^\star$
  - In Ethereum: $\mathbf{b}^\star$ **= 15M gas**
- Define a resource limit $\mathbf{b}$
  - Tx included must satisfy $\mathbf{Ax \le b}$

We constrain & charge for each resource

- Define a resource target $\mathbf{b}^\star$
  - Deviation from the target is $\mathbf{Ax} - \mathbf{b}^\star$
  - In Ethereum: $\mathbf{b}^\star$ **= 15M gas**
- Define a resource limit $\mathbf{b}$
  - Tx included must satisfy $\mathbf{Ax} \leq \mathbf{b}$
- Charge for each resource usage (EIP-1559)
  - $\mathbf{p}$ → transaction $\mathbf{j}$ costs $\mathbf{p}^T\mathbf{a}_j = \Sigma_i \mathbf{p}_i(\mathbf{a}_j)_i$

But how do we determine prices?

- We want a few properties:
  - $\mathbf{(Ax)_i} = \mathbf{b^\star_i} \rightarrow$ no update
  - $\mathbf{(Ax)_i} > \mathbf{b^\star_i} \rightarrow$ $\mathbf{p_i}$ increases
  - $\mathbf{(Ax)_i} < \mathbf{b^\star_i} \rightarrow$ $\mathbf{p_i}$ decreases

But how do we determine prices?

- We want a few properties:
  - $(Ax)_i = b^\star_i \rightarrow$ no update
  - $(Ax)_i > b^\star_i \rightarrow$ p_i increases
  - $(Ax)_i < b^\star_i \rightarrow$ p_i decreases
- Proposal

$$p_i^{k+1} = p_i^k \cdot \exp\left(\eta(Ax - b^\star)_i\right)$$

Is this a good update rule?

Update rules are implicitly solving an optimization problem.

Specific choice of objective by network designer → rule.

Section 2

The resource allocation problem

Setting (for now):

Network designer is omniscient & determines tx in each block.

Loss function is network designer's 'unhappiness' w. resource utilization

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ \infty & \text{otherwise.} \end{cases}$$

$$\ell(y) = \begin{cases} 0 & y \leq b^\star \\ \infty & \text{otherwise.} \end{cases}$$

Network designer determines **loss function** to define resource allocation problem

We encode all tx constraints in set S

- $\mathbf{S \subseteq \{0, 1\}^n}$ is the set of allowable transactions
  - Network constraints, e.g. $\mathbf{Ax \leq b}$
  - Interactions among tx's, e.g., bidders for MEV

We encode all tx constraints in set S

- $\mathbf{S} \subseteq \{0, 1\}^n$ is the set of allowable transactions
  - Network constraints, e.g. $\mathbf{Ax \leq b}$
  - Interactions among tx's, e.g., bidders for MEV
- Consider the convex hull of **S**: **conv(S)**
  - $\mathbf{x_j} \in (0,1) \Rightarrow$ tx **j** included after roughly $\mathbf{1/x_j}$ blocks

Transaction **j** included → user's & validators' joint utility is $\mathbf{q}_j$

- Tx producers = users + validators
- We almost never know **q** in practice
- However, the network does not need to know **q!**

Transaction producers get **utility** from each included transaction

The resource allocation problem:

$$
\begin{array}{ll}
\text{maximize} & q^T x - \ell(y) \\
\text{subject to} & y = Ax \\
& x \in \mathbf{conv}(S).
\end{array}
$$

- This is 'best case' scenario: tx's included to maximize utility, BUT
  - Cannot be implemented–designer does not build blocks!
  - **q** is unknowable
  - Cannot partially include tx's!

Section 3

Setting prices via duality

Duality theory: relaxing constraints to penalties

$$\begin{array}{ll} \text{maximize} & q^T x - \ell(y) \\ \text{subject to} & \boxed{y = Ax} \\ & x \in \mathbf{conv}(S). \end{array}$$

Relax this

- Network designer cares about throughput **y**, based on inc. tx's **x**
- We `decouple` utilization of network and that of tx producers
- Correctly set penalty → the *dual problem* is equivalent to the original problem & these utilizations are equal

Dual decouples tx producers & network

Network Problem **Block Building Problem**

$$g(p) = \sup_{y} \left(p^T y - \ell(y)\right) + \sup_{x \in \mathbf{conv}(S)} \left((q - A^T p)^T x\right).$$

- **p** is dual variable for constraint **y = Ax**
  - Relaxing constraint to penalty → pay per unit violation
- First term is easy to evaluate: can be done on chain!
- Let's look at the second term…

Second term: block building problem

$$\begin{array}{ll} \text{maximize} & (q - A^T p)^T x \\ \text{subject to} & x \in \mathbf{conv}(S), \end{array}$$

- Maximizing net tx utility subject to tx constraints
- Same optimal value as replacing conv(S) with S!
- Solved by block producers! → Network can observe $\mathbf{x}^\star$

What do we get at optimality?

- Let **p**★ be a minimizer of **g(p)** *[prices are set optimally]*
- Assume the block building problem has optimal sol **x**★
- The optimality conditions are

$$\nabla g(p^\star) = y^\star - Ax^\star = 0$$

- Where **y**★ satisfies $\nabla \mathcal{U}(\mathbf{y}^\star) = \mathbf{p}^\star$

Prices that minimize g charge the tx producers exactly the marginal costs faced by the network:

$$\nabla \ell(Ax^\star) = p^\star$$

And these prices incentivize tx producers to include tx's that maximize welfare generated $\mathbf{q}^T\mathbf{x}$ minus the network loss $\ell(\mathbf{y})$

Cool. So how do we minimize g(p)?

- We can compute the gradient:

$$\nabla g(p) = y^\star(p) - Ax^\star(p)$$

Cool. So how do we minimize g(p)?

- We can compute the gradient:

$$\nabla g(p) = y^\star(p) - Ax^\star(p)$$

- Network determines $\mathbf{y^\star(p)}$ (computationally easy)

Cool. So how do we minimize g(p)?

- We can compute the gradient:

$$\nabla g(p) = y^\star(p) - Ax^\star(p)$$

- Network determines $\mathbf{y^\star(p)}$ (computationally easy)
- $\mathbf{x^\star(p)}$ found by observing tx's included in block by tx producers

Cool. So how do we minimize g(p)?

- We can compute the gradient:

$$\nabla g(p) = y^\star(p) - Ax^\star(p)$$

- Network determines $\mathbf{y^\star(p)}$ (computationally easy)
- $\mathbf{x^\star(p)}$ found by observing tx's included in block by tx producers
- Then apply any optimization method (e.g., gradient descent)

$$p^{k+1} = p^k - \eta \nabla g(p^k).$$

Some simple examples:

Loss function

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ \infty & \text{otherwise.} \end{cases}$$

Update rule

$$p^{k+1} = p^k - \eta\left(b^\star - Ax^\star\right)$$

Some simple examples:

Loss function | **Update rule**

$$\ell(y) = \begin{cases} 0 & y = b^\star \\ \infty & \text{otherwise.} \end{cases}$$

$$p^{k+1} = p^k - \eta\left(b^\star - Ax^\star\right)$$

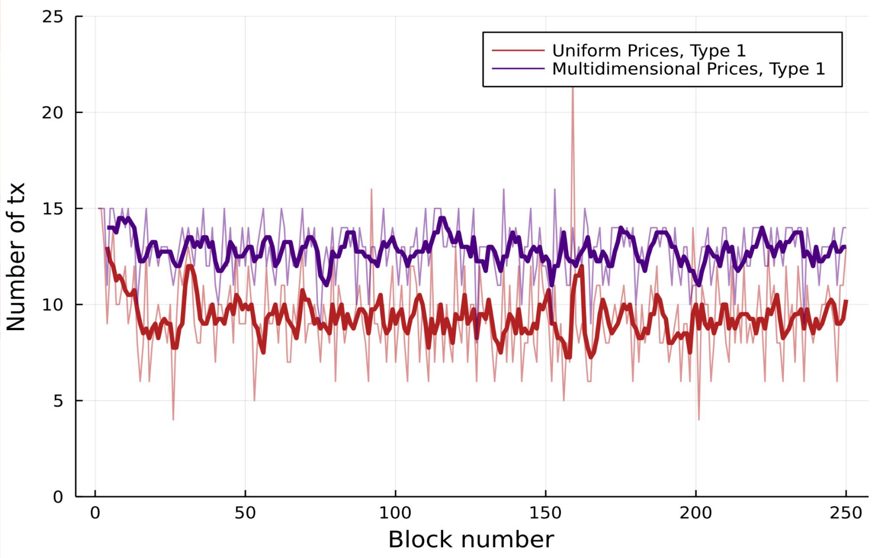
$$\ell(y) = \begin{cases} 0 & y \leq b^\star \\ \infty & \text{otherwise.} \end{cases}$$

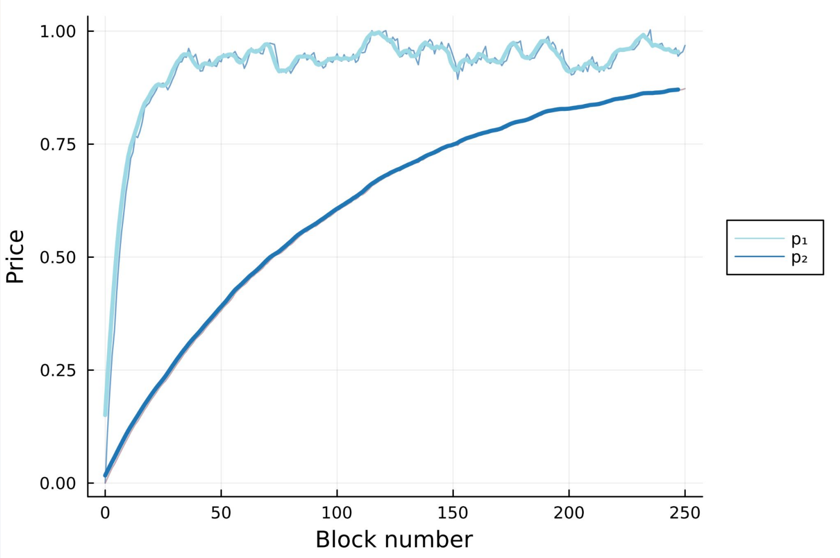
$$p^{k+1} = \left(p^k - \eta\left(b^\star - Ax^\star\right)\right)_+$$

Section 4

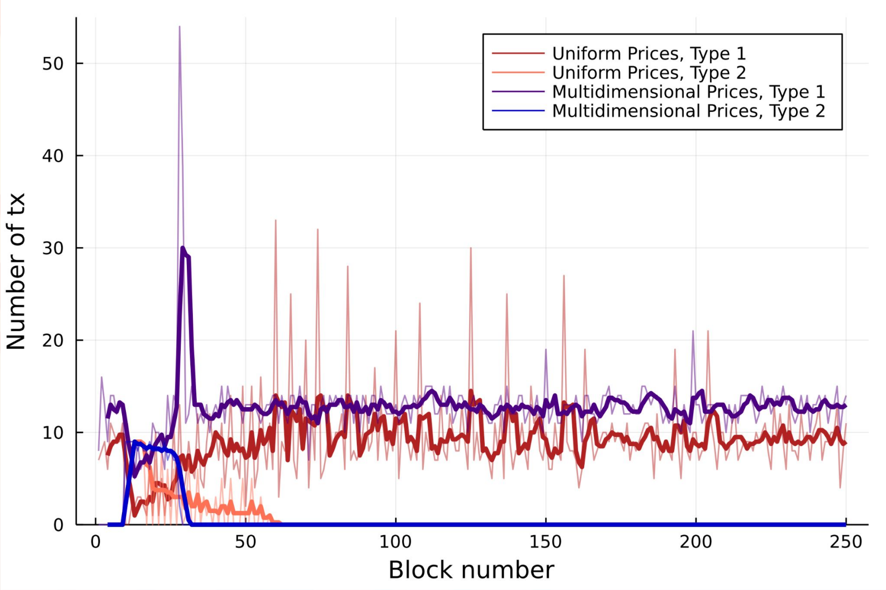
1-dim prices hurt networks

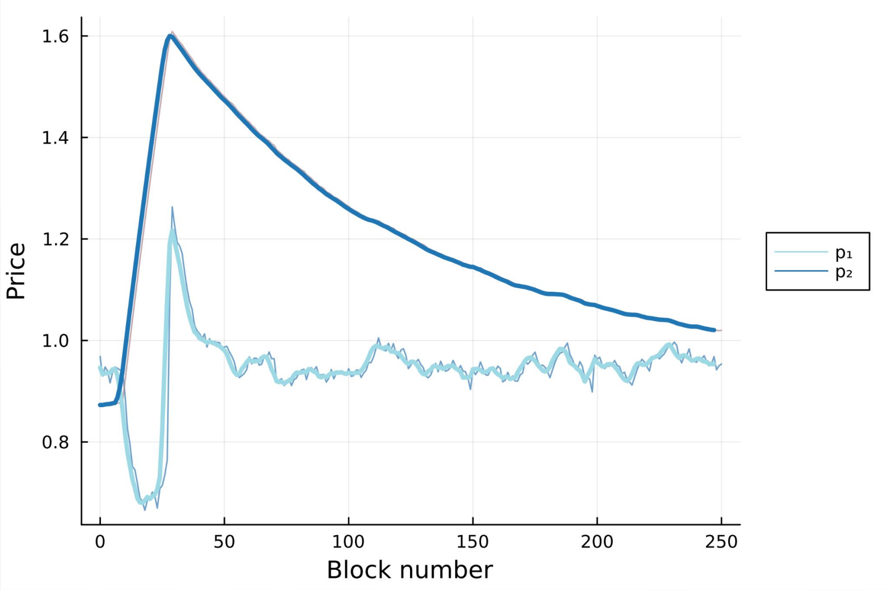
Multidimensional fees increase throughput

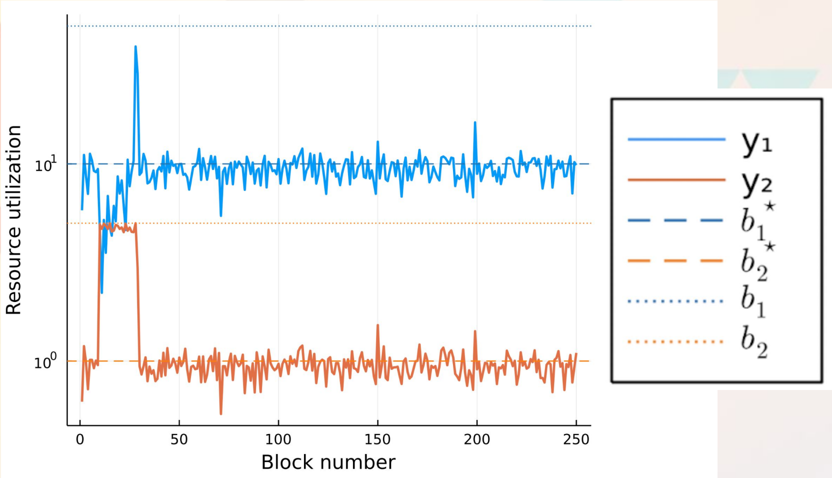


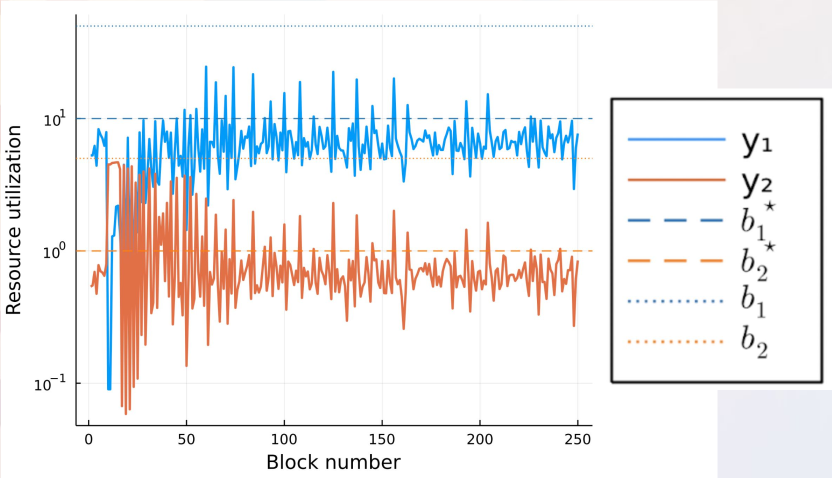


Even when the tx distribution shifts









And resource utilization better tracks targets

Future work & open questions

For researchers:

- What is the dynamical behavior? How do we make this strategy-proof? [Game-theoretic analysis of incentives]
- What update rules are most useful? [convergence behavior vs complexity]

For system designers:

- What should the resources be in a given system?
- How do you optimally trade-off between complexity & ease of use?
- How do you design a loss function for desired performance characteristics?

For more, check out our paper!

Thank you!

Theo Diamandis

Bain Capital Crypto & MIT

@theo_diamandis