Large Ethereum Contracts and Gas Fee Non-linearity

Maryam Bahrani, Miranda Christ, Daniel Jaroslawicz, Eric Neyman

December 12, 2020

1. Introduction

1.1. Background

One of the main motivations for Ethereum was to allow users to make arbitrary contracts, e.g. ones that resolve conditionally, or where the recipient depends on the state of the blockchain, rather than being limited to "A pays B amount X" as in Bitcoin. With the rise of decentralized finance (DeFi), such contracts, termed "smart contracts," are seeing increased popularity.

When a miner submits a block, the protocol requires that the miner check all contracts for validity. This requires computing the contract (i.e. running the contract's code), a task that may be computationally intensive. In order to incentivize miners to do these computations, rather than simply mining empty blocks, a *transaction fee* is paid from the author of the contract to the miner who includes the contract in their block. The amount of this fee is, roughly speaking, proportional to the intensity of the required computation. In Ethereum, computational intensity is measured in *gas* (roughly speaking, the number of steps required to run the computation). A transaction fee consists of a *gas limit* (units: gas; an upper bound on the gas required, reported by the contract author) and a *gas price* (units: ETH/gas; how much the author pays for every gas). The amount of the fee (i.e. the amount transferred from the contract author to the miner) is the product of these two numbers. The simplest contract (a transaction sending Ether from one account to another) costs about 21 thousand gas, and a typical gas price is 10^{-8} ETH/gas, so a typical transaction fee for a simple transaction is on the order of $2 \cdot 10^{-4}$ ETH, which is about 10 cents.

On the other hand, more complicated transactions are associated with both larger fees (because the gas limit is higher) and longer computation times. The structure of transaction fees in Ethereum makes it natural for the fee amount to be (linearly) proportional to the computation time (as both are proportional to the gas limit, assuming the gas price does not depend on the gas limit). This makes intuitive sense, as the opportunity cost for the miner of computing the contract is proportional to the time it takes to do so.

1.2. Motivation

The intuition that the transaction fee ought to be proportional to the time it takes to compute the contract breaks down for contracts whose computation depends on the state of the blockchain. For such contracts, if a block is mined while a different miner is computing the contract, the second miner's work is lost; the miner must start over. The chance of this happening is not necessarily proportional to the time that it takes to compute the contract. As we shall see, under some assumptions this causes the opportunity cost of computing the contract to be sublinear in the computation time; under others, it is superlinear. As a result, contract authors may wish to set their gas prices as a function of the computational intensity of their transaction, instead of setting them to some constant value.

With Ethereum as it exists today, we believe that these effects are fairly inconsequential because the block gas limit (i.e. limits on how much gas worth of contract can go in a block) is such that computing all contracts is relatively inexpensive compared to mining blocks. It is worth noting however that while the cost of computationally expensive contracts is very small compared to the cost of mining, more memory-intensive contracts can have higher (potentially non-negligible) costs even with current parameters. Furthermore, as Ethereum becomes more popular, there is a push toward increasing the block gas limit. For example, there are already complaints about prolonged congestion periods in the Ethereum network.[1] Proposals for new transaction fee mechanisms such as EIP-1559 also involve doubling the block gas limit. In the long run, if the block gas limit is raised substantially, some transactions may become expensive enough that the aforementioned nonlinear effects become consequential. The goal of this work is to explore these effects.

1.3. Model

Below is a list of our modeling assumptions, along with some discussion of how realistic they are and their implications about the reach of our result.

myopic miners We focus on the time scale of one block, and assume miners maximize their expected reward (both block reward and transaction fees) from the next block only. A more natural model for miners' objective is maximizing revenue over time. The total revenue from transactions is simply the sum of transaction fees in their valid blocks, whereas the total revenue from block rewards can vary depending the total hashrate devoted to mining due to difficulty adjustment. In other words, since difficulty adjustment ensures that the amount of coins generated over a long interval is fixed regardless of the total mining hashrate, a miner's revenue from block rewards is maximized by maximizing her fraction of this fixed amount, which is equivalent to maximizing her fraction of the mining hashrate. In particular, in choosing to allocate some computational power to computing a transaction instead of mining, a miner decreases both her mining hashrate as well as the overall mining hashrate, compensating for some of the miner's revenue loss from block rewards. This compensation effect is larger for larger miners, so our assumption that miners are myopic underestimates large miners' willingness to compute transactions instead of mining. Our results show that large miners are more inclined to compute large transactions *despite this*, making our result stronger.

extreme state-dependence We assume contracts crucially depend on the state of the blockchain and must be recomputed from scratch every time a new block is added. In general, detecting whether a piece of code has this property faster than running it could be hard or impossible (think of the halting problem), but we expect our results to apply (albeit to a lesser extent) to any contract that poses a risk of being state dependent.

[1] `https://forkast.news/ethereum-defi-transactions-congestion-scaling-ponzi-report/`

synchronized start We fix a mempool state and a set of miners, and assume all miners have access to this mempool and start working on the next block at the same time. In practice, this assumption might not hold due to network delays, which might systematically favor some groups of miners. Additionally, when a new block is published, most miners presumably verify it to ensure its validity and update state as necessary, a process which likely takes longer for smaller miners. Since large miners are more likely to systematically benefit from these effects, our results is only stronger for not taking them into consideration. For the majority of this report (except for Section 3.1), we further assume that the mempool consists of only one contract.

honest verification We assume miners only include contracts that they have computed. This is the intended behavior by the protocol, and is reasonable to assume for rational miners and small transactions with low fees, where the cost of computing the transaction is small relative to the gain from expected block reward eliminating the risk of mining an invalid block. This breaks down for, say, a large transaction with high fee that looks likely to be valid, since a miner might prefer to stomach the small risk of mining an invalid block in the hopes of gaining the large gas fees from being first to mine. This scenario is further complicated by the fact that a block containing a contract with very high total gas fee looks very attractive for fee sniping attacks, making the large fee seem less appealing when factoring in the chances of its inclusion causing the whole block to be orphaned.

The dilemma faced by the miners about whether to actually run a contract or simply claim they have done so is an example of *verifier's dilemma*. Another example of verifier's dilemma arises when miners prefer not to spend resources on verifying the validity of contracts already on the chain and rely on other miners to do so. Although not the focus of this report, the verifier's dilemma is an interesting topic in its own right with many open questions (see Luu *et al.*[2] for a discussion, and Pontiveros *et al.*.[3] for a related attack).

single-tasking We assume (except in section 4.3, where we make an empirical argument for the validity of this assumption) that miners only attempt to mine after fully executing (or deeming invalid) all contracts they intend to include. In practice, it might be beneficial to only spend a fraction of computational resources on running contracts and use the rest to attempt mining a block with only transactions that have finished executing (allocating cycles back to mining as they free up upon finishing transactions). The correct allocation of resources depends on how effective a miner's computational resources are at running contracts relative to mining, since in the case of extremely specialized hardware, there is a dominant allocation respecting that specialization. Furthermore, the degree of parallelizability of a contract affects the rate of diminishing returns from allocating cycles to it. One argument in favor of our assumption is the prevalence of mining pools. Keeping participants informed about the state of all running transactions can be difficult and hindered by network delays. Mining pools generally decide on a set of contracts in a greedy fashion and ask each par-

[2]Demystifying Incentives in the Consensus Computer, Luu, Teutsch, Kulkarni, Saxena 2015, `https://eprint.iacr.org/2015/702.pdf`

[3]Sluggish Mining: Profiting from the Verifier's Dilemma, Pontiveros, Torres, State 2019, `https://fc19.ifca.ai/wtsc/SluggishMining.pdf`

ticipant to run them separately. The high cost of coordination might make an otherwise profitable plan not worthwhile for mining pools.

running time We work with a few different models about how long it takes each miner to compute a given transaction. In Section 3, we consider a contract that takes time t to calculate for every miner regardless of hashrate, where t is known in advance to all miners. In Section 4, we assume calculating a contract takes miners time inversely proportional to their hashrate, and the hash rates of all miners are publicly known in advance. Finally, in Section 5, we give a more general expression that makes no assumptions about the running time other than the fact that its value for each miner is publicly known. We give a more detailed description and a discussion of applicability of each model in its corresponding section.

2. Results

We start in section 3 by examining gas prices in the *uniform time setting*, where the time it takes to compute a transaction is the same for all miners. In this setting, we derive an expression for the minimum fee that Alice needs to be paid in order for a transaction to be worth including in her block. We also prove that if a miner with hashrate α_1 prefers not to do a computation, a smaller miner with hashrate $\alpha_2 \leq \alpha_1$ also prefers not to do the computation. We also discuss equilibria and analyze in detail the case where there are only two miners. However, we recognize that the uniform time assumption yields an idealized model; in reality, more powerful miners can perform some computations faster than less powerful miners.

In section 4, we extend our analysis to the *proportional time setting*, where the time required for a miner to compute a transaction is inversely proportional to the miner's hashrate. In this section, we show first that if no miners other than Alice compute the transaction, the minimum transaction fee that motivates her to include the transaction is $e^{t/\alpha - t} - 1$ (Claim 5). We then derive a lower bound on the minimum transaction fee when Alice has the largest hashrate of any miner, and some of the other miners do compute the transaction. Finally, we consider the scenario where a large miner can allocate some of their hashrate to mining empty blocks while the rest of their hashrate is used for computing a transaction. We discuss the optimal allocation and provide empirical evidence that it is often most profitable to either mine with full power or to compute with full power.

In section 5, we derive an expression for gas prices in a generalized time model. We make no assumption about the amount of time it takes a miner to compute a transaction other than the fact that its value for each miner is publicly known to all miners in advance. We show that our new expression matches the corresponding result in the uniform time setting.

3. Gas prices and equilibria in the uniform time setting

Suppose that there is a smart contract in the mempool that takes time t to calculate, a value that does not depend on the miner doing the computation and is known in advance to all miners. The units of t are "multiples of the average time between blocks if everyone is mining from the beginning." (So for instance, $t = 0.1$ would imply that if the average time between blocks is 10 seconds, then the transaction takes one second to compute.)

The assumption that the computation takes every miner the same amount of time is dubious in many settings, but is perhaps reasonable if the computation is not easily parallelizable. Additionally, it makes the math substantially simpler. We call this setting the *uniform time setting.* Furthermore, we believe the assumption that miners know t in advance is fairly reasonable, because in many cases miners will encounter contracts of a form that they have seen many times before, in which case they will know roughly how long the computation will take. Furthermore, the contract gas limit can be used as a reasonable predictor of t, since (as we argue below) per-op gas price is monotone increasing in t, so it is in the best interest of the author of a contract to provide as tight an upper bound on t as possible.

Consider a miner Alice with hashrate α. Let p be the total hashrate of all miners (not including Alice) who choose to *not* do the computation (meaning that they are mining from the beginning). (Note that the value of p is different for different miners.) Without loss of generality, say that the block reward is 1. We wish to know the amount that Alice needs to be paid (as a function of t) for including the transaction in her block so that computing the transaction is worthwhile to Alice. We will denote this amount $g_{\alpha,p}(t)$.

Claim 1. In the uniform time setting, Alice needs to be paid

$$g_{\alpha,p}(t) = e^{pt}\left(\frac{1}{\alpha+p}(1-e^{-(\alpha+p)t}) + e^{-(\alpha+p)t}\right) - 1$$

for including the contract in her block in order for the contract to be worth her time to compute.

Proof. First suppose that Alice decides not to do the computation. What is her expected reward? Well, it is equal to the probability of her mining the next block. There are two phases to consider: the one where only the miners who do not do the computation are mining (these have a hashrate of $\frac{\alpha}{\alpha+p}$) and the one where all miners are mining.

Conditional on a block being mined during the first phase, there is a $\frac{\alpha}{\alpha+p}$ chance that the block is mined by Alice. The probability that the block is mined during this period is $1 - e^{-(\alpha+p)t}$. This is the CDF of the exponential distribution with parameter $\alpha + p$ (the hashrate of all miners during the first phase), evaluated at t. If a block is not mined during this phase, there is an α chance that the block is mined by Alice. Therefore, if Alice does not do the computation, her expected reward is

$$\frac{\alpha}{\alpha+p}(1-e^{-(\alpha+p)t}) + \alpha e^{-(\alpha+p)t}.$$

Now suppose that Alice decides to do the computation, and say that the total amount paid to a miner for including the transaction is g. Then her reward conditional on mining the next block is no longer 1 but rather $1 + g$. The probability that she mines the block is α times the probability that no one mines the block during the phase where some miners are working on the computation. This probability is e^{-pt}. Therefore, Alice's expected reward if she does the computation is

$$(1+g)\alpha e^{-pt}.$$

Therefore, it is worthwhile for Alice to do the computation if and only if

$$(1+g)\alpha e^{-pt} \geq \frac{\alpha}{\alpha+p}(1-e^{-(\alpha+p)t}) + \alpha e^{-(\alpha+p)t}.$$

Therefore, we have

$$g_{\alpha,p}(t) = e^{pt}\left(\frac{1}{\alpha+p}(1-e^{-(\alpha+p)t}) + e^{-(\alpha+p)t}\right) - 1,$$

as desired. $\square$

Let us consider a few special cases of Claim 1. The first case is that α **is small**. We have

$$g_{0,p}(t) = e^{pt}\left(\frac{1}{p}(1-e^{-pt}) + e^{-pt}\right) - 1 = \frac{1}{p}(e^{pt}-1).$$

Observe that this function is exponential in t. If we assume that t is very small then this does not matter: we may approximate $e^{pt} \approx 1 + pt$ and obtain $g_{\alpha,p}(t) \approx t$. However, we are interested in situations where the computation time is non-negligible, and in such situations we have found that if a large fraction of miners do not do the computation, it makes sense for the party that sends the computation to the mempool to pay a gas price that is increasing (exponentially, in fact) as a function of the complexity of the computation (or equivalently, the gas limit).

The second case is that p **is small**, i.e. almost everyone does the computation. This is the way the protocol is intended to work. We have

$$g_{\alpha,0}(t) = \frac{1}{\alpha}(1-e^{-\alpha t}) + e^{-\alpha t} - 1 = \left(\frac{1}{\alpha} - 1\right)(1-e^{-\alpha t}).$$

Note that this quantity decreases in α:

$$\begin{aligned}\frac{d}{d\alpha}g_{\alpha,0}(t) &= \frac{-1}{\alpha^2}(1-e^{-\alpha t}) + \left(\frac{1}{\alpha}-1\right)te^{-\alpha t} = \left(\frac{1}{\alpha^2} + \frac{t}{\alpha} - t\right)e^{-\alpha t} - \frac{1}{\alpha^2} \\ &= \frac{e^{-\alpha t}}{\alpha^2}\left(1 + \alpha t - \alpha^2 t - e^{\alpha t}\right) \leq \frac{e^{-\alpha t}}{\alpha^2}\left(1 + \alpha t - \alpha^2 t - 1 - \alpha t\right) = -te^{-\alpha t} < 0.\end{aligned}$$

This means one needs to pay a miner with a larger hashrate less to incentivize them to compute a contract. Put otherwise, if p is small (which is true if the protocol is working well), we should expect the miners who do not do the computation to be the ones with the smallest hashrates. As we will later show, this is true even if p is not small.

Note also that unlike the $\alpha \to 0$ setting, where the price of doing a computation grew exponentially in t, here it is in fact *bounded*, having a limit of $\frac{1}{\alpha} - 1$ as $t \to \infty$. This limiting behavior makes sense: if everyone else does the computation, then the miner can be almost guaranteed to mine the next block (have reward 1) or do the computation and compete with the other miners for the block (get reward $1+g$ with probability α). Solving for the value of g where the two expected rewards are the same gives us $g = \frac{1}{\alpha} - 1$.

Finally, we consider a third limiting case, where α **and** p **are similarly small**. Let $p = c\alpha$. We have

$$\lim_{\alpha\to 0} g_{\alpha,c\alpha}(t) = \lim_{\alpha\to 0} e^{c\alpha t}\left(\frac{1}{(c+1)\alpha}(1-e^{-(c+1)\alpha t}) + e^{-(c+1)\alpha t}\right) - 1 = \lim_{\alpha\to 0}\frac{1-e^{-(c+1)\alpha t}}{(c+1)\alpha} = t,$$

where we apply L'Hôpital's rule in the last step.

To summarize, in the limit as $\alpha \to 0$, the cost of doing a computation is exponential in t. In the limit as $p \to 0$, the cost is bounded as $t \to \infty$. In the case where we send α and p to zero at the same rate, the cost grows linearly in t. This is, in some sense, the desired behavior, as the gas price (i.e. price for doing one unit of computation) is often treated as a constant independent of the computation.

We also consider one more special case that will be useful later, which is when **no one else does the computation**, i.e. $p = 1-\alpha$. We have

$$g_{\alpha,1-\alpha}(t) = e^{(1-\alpha)t}(1-e^{-t}+e^{-t}) - 1 = e^{(1-\alpha)t} - 1.$$

Recall that we showed that when $p = 0$, the higher a miner's hashrate the more sense it makes for the miner to do the computation. We now show this in more generality.

Claim 2. Suppose that in the uniform time setting, a miner with hashrate α_1 prefers not to do a computation. Assuming that this miner does not do the computation, a miner with hashrate $\alpha_2 \le \alpha_1$ also prefers not to do the computation.

Proof. Suppose for contradiction that the miner with hashrate α_2 prefers to do the computation. Let p_1 be the value of p for the first miner and p_2 be the value of p for the second miner (so $p_2 = p_1 + \alpha_1$, since the first miner does not do the computation and the second miner does). By assumption, we have that $g_{\alpha_1,p_1}(t) > g_{\alpha_2,p_2}(t)$. From Claim 1, it follows that

$$e^{p_1 t}\left(\frac{1}{\alpha_1+p_1}(1-e^{-(\alpha_1+p_1)t}) + e^{-(\alpha_1+p_1)t}\right) - 1 > e^{p_2 t}\left(\frac{1}{\alpha_2+p_2}(1-e^{-(\alpha_2+p_2)t}) + e^{-(\alpha_2+p_2)t}\right) - 1$$
$$\frac{1}{\alpha_1+p_1}(1-e^{-(\alpha_1+p_1)t}) + e^{-(\alpha_1+p_1)t} > e^{\alpha_1 t}\left(\frac{1}{\alpha_2+p_2}(1-e^{-(\alpha_2+p_2)t}) + e^{-(\alpha_2+p_2)t}\right).$$

Let $x_i := \alpha_i + p_i$, so $x_1 = x_2 - \alpha_2$. Then

$$\begin{aligned}\frac{1}{x_1}(1-e^{-x_1 t}) + e^{-x_1 t} > e^{\alpha_1 t}\left(\frac{1}{x_2}(1-e^{-x_2 t}) + e^{-x_2 t}\right) &\ge e^{\alpha_2 t}\left(\frac{1}{x_2}(1-e^{-x_2 t}) + e^{-x_2 t}\right)\\ &= \frac{e^{\alpha_2 t}}{x_2} + \left(\frac{-1}{x_2}+1\right)e^{-x_1 t}\\ \frac{1}{x_1} - \frac{e^{\alpha_2 t}}{x_2} &> \left(\frac{1}{x_1}-\frac{1}{x_2}\right)e^{-x_1 t}\\ x_2 - x_1 e^{\alpha_2 t} &> (x_2-x_1)e^{-x_1 t}.\end{aligned}$$

Now, using the fact that $e^x \ge 1+x$ on both $e^{\alpha_2 t}$ and $e^{-x_1 t}$, we have

$$x_2 - x_1(1+\alpha_2 t) > (x_2-x_1)(1-x_1 t)$$

$$\begin{aligned} x_2 - x_1 - x_1\alpha_2 t &> x_2 - x_1 - x_1 x_2 t + x_1^2 t \\ x_1 t(x_2 - \alpha_2 - x_1) &> 0. \end{aligned}$$

But this is a contradiction, as $x_2 = \alpha_2 + x_1$. This completes the proof. $\square$

We observe the following corollary about equilibria in the uniform time setting.

Corollary 3. In the uniform time setting, any pure equilibrium satisfies the property that for some α (depending on t and the reward g for doing the computation), all miners above hashrate α compute the contract and all miners below hashrate α do not.

Proof. Suppose for contradiction that this is not the case. Then there is an equilibrium in which a miner with hashrate α_1 does not do the computation, but a miner with hashrate $\alpha_2 < \alpha_1$ does. But this is impossible, by Claim 2. $\square$

We now consider a special case, namely that of two miners. Say that miner 1 has hashrate $\alpha > 0.5$ (so the other miner has hashrate $1 - \alpha$). The following table summarizes how large the reward g for including the contract in the block needs to be in order for each miner to do the computation, under varying assumptions about what the other miner does.

Miner #	Other miner computes	Other miner does not
1	$g_{\alpha,0}(t) = \frac{1-\alpha}{\alpha}(1 - e^{-\alpha t})$	$g_{\alpha,1-\alpha}(t) = e^{(1-\alpha)t} - 1$
2	$g_{1-\alpha,0}(t) = \frac{\alpha}{1-\alpha}(1 - e^{-(1-\alpha)t})$	$g_{1-\alpha,\alpha}(t) = e^{\alpha t} - 1$

The table below summarizes the requirements on g to achieve each hypothetical pure equilibrium.

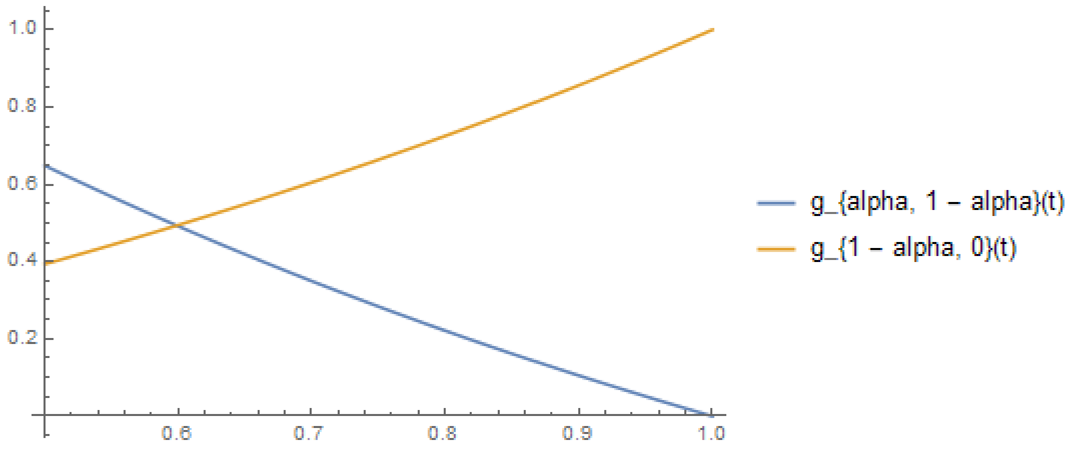
	Miner 2 computes	Miner 2 does not
Miner 1 computes	$g \geq \max(g_{\alpha,0}(t), g_{1-\alpha,0}(t)) = g_{1-\alpha,0}(t)$	$g_{\alpha,1-\alpha}(t) \leq g \leq g_{1-\alpha,0}(t)$
Miner 1 does not	Impossible (Corollary 3)	$g \leq \min(g_{\alpha,1-\alpha}(t), g_{1-\alpha,\alpha}(t)) = g_{\alpha,1-\alpha}(t)$

The equality in the top-left cell follows from the fact that $g_{\alpha,0}(t)$ is a decreasing function of α (as we showed earlier). The inequality in the bottom-right cell follows from the fact that $e^{(1-\alpha)t} \leq e^{\alpha t}$.

This means that:

- If $g_{\alpha,1-\alpha}(t) < g_{1-\alpha,0}(t)$, then exactly one equilibrium is achievable, depending on g.
- If $g_{\alpha,1-\alpha}(t) = g_{1-\alpha,0}(t)$, then either both miners compute or neither does, unless g is equal to this value, in which there are three pure equilibria.
- If $g_{\alpha,1-\alpha}(t) > g_{1-\alpha,0}(t)$, then if g is large or small then there is exactly one equilibrium (both miners compute or both don't, respectively), and if g is intermediate in size (between $g_{1-\alpha,0}(t)$ and $g_{\alpha,1-\alpha}(t)$ then *both* of these are equilibria, but there is no equilibrium in which one miner computes and the other does not.

As an example, consider the case of $t = 1$.



If $\alpha \geq 0.599$, we have that $g_{\alpha,1-\alpha}(t) < g_{1-\alpha,0}(t)$, so there is a range of values of g for which miner 1 does the computation but miner 2 does not. On the other hand, if $\alpha \leq 0.598$, no such g exists. Instead, there is a range of values of g for which "both miners compute" and "neither miner computes" are both equilibria.

3.1. Selecting a revenue maximizing transaction

In the above analysis, we considered a single contract and found the minimum gas price required in order for that contract to be worth a miner's time to compute. A natural extension is to ask: if there are multiple outstanding transactions in the mempool, which contract is most valuable for the miner to compute?

In this section, we attempt to answer the above question. This is incomplete at the moment and we conjecture that it is true, though it should be verified.

Specifically, consider a miner Alice with hashrate α. As defined above, let p be the total hashrate of all miners (not including Alice) who choose to do *no* computation (meaning that they are mining from the beginning). Let there be a set $S = \{(g_1, t_1), (g_2, t_2)\}$ of two outstanding transactions in the mempool, where t_i is the time it takes to compute transaction i and g_i is the total reward collected by a miner for including the transaction in a block. Let α_1 be the total hashrate of all miners who choose to compute transaction 1 and let α_2 be the total hashrate of all miners who choose to compute transaction 2. In essence, we are considering a more general version of the single contract, uniform time model analyzed above where Alice can now choose between computing any one of two transactions instead of between one real transaction and one empty transaction (i.e. starting to mine immediately on the block). We wish to know the amount that Alice needs to be paid (as a function of (t_1, t_2)) to choose the transaction with a longer required compute time. We will denote this amount $g_{g_2,\alpha,\alpha_1,\alpha_2,p}(t_1, t_2) = g_1$.

Claim 4. In the uniform time setting, where, without loss of generality, $t_1 \geq t_2$, Alice must be paid at least:

$$g_1 = (1 + g_2)e^{(p+\alpha_2)(t_1-t_2)} \left(\frac{1}{\alpha + \alpha_2 + p}(1 - e^{-(\alpha+\alpha_2+p)(t_1-t_2)}) + e^{-(\alpha+\alpha_2+p)(t_1-t_2)} \right) - 1$$

.

in order for her to compute the longer transaction.

Proof. Since Alice is computing transaction i she will receive reward $(1+g_i)$ if she successfully mines the block. There are three phases to consider when calculating her expected reward: the one where only the miners who do not do the computation are mining (these have a hashrate of p), the one where Alice finishes her computation but no other miners doing the full computation have completed theirs yet (so Alice has effective hashrate $\frac{\alpha}{\alpha+p}$ during this period), and the one where all miners are mining (where Alice has hashrate α).

Conditional on a block not being mined during the first phase and being mined during the second phase, there is a $\frac{\alpha}{\alpha+p}$ chance that the block is mined by Alice. The probability that the block is mined during this phase and not mined during the first phase is $e^{-pt_i} \cdot (1 - e^{-(\alpha+p)(T-t_i)})$. Similar to our analysis in the above uniform time setting, we find this probability using the CDF of the exponential distribution with parameter set equal to the effective hashrate during the phase under consideration. If a block is not mined during this phase, there is an α chance that the block is mined by Alice. Therefore, Alice's total expected reward is

$$(1+g_i) \cdot \left(e^{-pt_i} \frac{\alpha}{\alpha+p} (1 - e^{-(\alpha+p)(T-t_i)}) + \alpha e^{-pt_i} e^{-(\alpha+p)(T-t_i)} \right).$$

Simplifying yields:

$$(1+g_i) \cdot \alpha \left(\frac{e^{-pt_i}}{\alpha+p} \left(1 + (\alpha+p-1) \cdot e^{-(\alpha+p)(T-t_i)}\right) \right).$$

We then take the argmax of this expression over the transaction set to find the transaction maximizing Alice's expected reward, yielding the expression in Claim 4. □

Note that this is consistent with our Claim 1 where we found the the minimum gas price required for a transaction to incentivize Alice to compute it. If $(g_2 = 0, t_2 = 0)$ (and p and α_2 are merged) then we have

$$g_1 = e^{pt} \left(\frac{1}{\alpha+p} (1 - e^{-(\alpha+p)t}) + e^{-(\alpha+p)t} \right) - 1$$

which is identical to what was shown above.

Special cases to consider are that $\frac{d^2(dp)}{dt_1^2} > 0$ implying that as p increases the rate at which minimum required g_1 grows is increasing with t_i. I.e. if everyone is working on the block then I need more incentive to spend a longer time computing

Another special case is that if p goes to $1-\alpha$ then the derivative of g_1 wrt to alpha is positive; i.e. a larger miner needs greater incentive to work on the heavier compute tx. This is interesting since it goes against the grain we've been seeing.

4. Gas prices in the proportional time setting

In the previous section, we assumed that the time each transaction takes to calculate is the same for every miner. While this model is convenient to analyze, it is not always aligned with reality. Consider, for example, a completely parallelizable transaction that takes t operations to compute. Suppose a small miner can perform α operations per second and a larger miner can perform $k\alpha$ operations per second (e.g. this is the case when miners have fully general-purpose hardware that is equally as effective at mining and at running transactions). The smaller miner will compute the transaction in $\frac{t}{\alpha}$ seconds, while the larger miner will compute the transaction in $\frac{t}{k\alpha}$ seconds. Therefore, with completely parallelizable transactions, the time it takes for a miner to compute a transaction is inversely proportional to the miner's hashrate. We call this the *proportional time setting.*

Note that this setting isn't entirely realistic either, since we measure computation in "operations," which we treat as identical. In reality, operation a might take twice as long as operation b. We can fix this by counting operation a as two operations and operation b as one operation. However, problems arise when an operation is longer *and* not parallelizable, in which case it cannot be treated as multiple smaller operations. Due to the magnitude of the total compute power devoted to mining Ethereum, this effect should be small, and it is reasonable to assume that the time it takes a miner to compute a transaction is inversely proportional to that miner's hashrate.

In the proportional time setting, we assume there is a smart contract in the mempool that takes a miner with hashrate α time $\frac{t}{\alpha}$ to calculate, a value that *does* depend on the miner doing the computation and is known in advance to all miners. We assume that the hashrate of every miner is publicly known. The units of time are again "multiples of the average time between blocks if everyone is mining from the beginning."

Consider a miner Alice with hashrate α. We wish to know under what circumstances Alice chooses to compute the transaction to include in her next block.

4.1. Only Alice computes

We first consider the simplified scenario where all miners other than Alice work exclusively on mining.

Claim 5. In the proportional time setting, assuming that all other miners work on mining (rather than computing transactions), Alice must be paid at least:

$$g = e^{t/\alpha - t} - 1$$

in order to include a transaction that takes her time $\frac{t}{\alpha}$ to compute.

Proof. As before, we can compare Alice's expected reward from the next block when she *does* perform the computation to her expected reward from the next block when she *does not* perform the computation. Alice only profits from including the transaction if her expected reward from the next block is higher when she does the computation than when she doesn't.

We first compute Alice's expected reward assuming that no other miners compute the transaction:

$$r = \begin{cases} \alpha & \text{if Alice does not compute tx} \\ \alpha(1+g) \cdot Pr[\text{no block is mined while Alice is computing}] & \text{if Alice does compute tx} \end{cases}$$

where r is Alice's expected reward from the next block and g is the transaction fee associated with the transaction. If Alice does not compute the transaction, her expected reward from the next block is 1 times the probability that she mines the next block, which is α. If Alice does compute the transaction, her expected reward from the next block is the reward including the transaction fee times the probability that she mines the next block. The probability that she mines the next block is the probability that no block is mined while Alice is computing times the probability that the next block mined is hers. Since the transaction takes Alice time $\frac{t}{\alpha}$ to compute, and all other miners mine while Alice is computing,

$$Pr[\text{no block is mined while Alice is computing}] = e^{-(1-\alpha)t/\alpha}$$

Assuming no other miners compute the transaction, Alice only benefits from computing the transaction if it increases her expected reward compared to when she only mines and does not compute. Therefore, Alice chooses to compute the transaction only if

$$\alpha(1+g)e^{-(1-\alpha)t/\alpha} \geq \alpha$$

$$1+g \geq e^{(1-\alpha)t/\alpha}$$

$$g \geq e^{t/\alpha - t} - 1$$

□

Observe that this quantity decreases with α. Therefore, it is cheaper to convince a larger hashrate miner than a smaller hashrate miner to work on a transaction, assuming all other miners do not work on the transaction.

If we want to set the minimum transaction fee such that *some* miner computes the transaction, it is reasonable to assume that no miners other than the largest miner compute the transaction. Under this assumption, the transaction fee for a transaction requiring t operations to compute $g(t) = e^{t/\alpha_{max} - t} - 1$, where α_{max} is the greatest hashrate of any miner.

4.2. Alice and some other miners compute

We extend the above analysis to the scenario where some of the other miners compute the transaction, rather than all mining empty blocks. Let the set of miners (not including Alice) who do not compute the transaction have combined hashrate p. In this section, we assume for simplicity that Alice's hashrate α is the largest of any miner.

Claim 6. In the proportional time setting, Alice needs to be paid a transaction fee of at least

$$g = e^{pt/\alpha} - e^{-t} + (\alpha + p)e^{-t} - 1$$

in order for including a transaction requiring time $\frac{t}{\alpha}$ to be more profitable than mining an empty block.

Proof. Let r be Alice's expected reward if she does not compute the transaction and let r_{tx} be Alice's expected reward if she does compute the transaction. We calculate a lower bound on r and an upper bound on r_{tx}. In order for computing the transaction to be profitable for Alice, the transaction fee must be large enough so that the upper bound on r_{tx} is at least the lower bound on r.

First, consider the case where Alice does not compute the transaction. While the $1-\alpha-p$ other miners compute the transaction, Alice's effective hashrate is $\frac{\alpha}{\alpha+p}$. As other miners finish the computation and resume mining, Alice's effective hashrate decreases. After all miners finish computing the transaction, Alice's effective hashrate returns to α. Observe that since $\alpha + p \leq 1$, $\frac{\alpha}{\alpha+p} \leq \alpha$. Therefore, in the period between, when some miners have finished computing and others are still computing, Alice's effective hashrate is at least α.

Let $\mathcal{E}$ be the event that a block is mined before any miner has finished the computation. Alice's expected reward when not computing the transaction is at most

$$r \geq \frac{\alpha}{\alpha+p} Pr[\mathcal{E}] + \alpha(1 - Pr[\mathcal{E}])$$

Since Alice has the largest hashrate of any miner, the time it takes any miner to compute the transaction is at least $\frac{t}{\alpha}$. Therefore, $Pr[\mathcal{E}] = 1 - e^{-(\alpha+p)t/\alpha}$. Plugging in this expression, we have

$$r \geq \frac{\alpha}{\alpha+p}(1 - e^{-(\alpha+p)t/\alpha}) + \alpha e^{-(\alpha+p)t/\alpha}$$

We now calculate Alice's expected reward r_{tx} when she does compute the transaction. When Alice finishes computing the transaction, she and at least the p other miners are mining. Therefore, her effective hashrate is at most $\frac{\alpha}{\alpha+p}$

$$r_{tx} \leq \frac{\alpha}{\alpha+p}(1+g) \cdot Pr[\text{nobody mines a block while Alice is computing}]$$

Since Alice computes for time $\frac{t}{\alpha}$, $Pr[\text{nobody mines a block while Alice is computing}] = e^{-pt/\alpha}$. Therefore,

$$r_{tx} \leq \frac{\alpha}{\alpha+p}(1+g)e^{-pt/\alpha}$$

If $r_{tx} \geq r$, then we must have

$$\frac{\alpha}{\alpha+p}(1+g)e^{-pt/\alpha} \geq \frac{\alpha}{\alpha+p}(1 - e^{-(\alpha+p)t/\alpha}) + \alpha e^{-(\alpha+p)t/\alpha}$$

We can set the sides equal and solve for g:

$$(1+g)e^{-pt/\alpha} = 1 - e^{-(\alpha+p)t/\alpha} + (\alpha+p)e^{-(\alpha+p)t/\alpha}$$

$$g = e^{pt/\alpha} - e^{-t} + (\alpha+p)e^{-t} - 1$$

Therefore, in order for Alice's expected reward to be higher when computing the transaction than when mining an empty block, we must have $g \geq g_{\alpha,p}(t)$, where

$$g_{\alpha,p}(t) := e^{pt/\alpha} - e^{-t} + (\alpha+p)e^{-t} - 1$$

□

Although we lower bound r and upper bound r_{tx}, our expression for $g_{\alpha,p}(t)$ is still reasonably close to the minimum transaction fee that motivates Alice to compute the transaction. In fact, for $p = 1 - \alpha$, setting g equal to $g_{\alpha,p}(t)$ makes Alice's reward from computing the transaction exactly equal her reward from not computing the transaction.

$$\begin{aligned} g_{\alpha,1-\alpha}(t) &= e^{(1-\alpha)t/\alpha} - e^{-t} + e^{-t} - 1 \\ &= e^{(1-\alpha)t/\alpha} - 1 \end{aligned}$$

Recall that Alice's expected reward from computing a transaction with fee g assuming that no one else computes the transaction is $\alpha(1+g)e^{-(1-\alpha)t/\alpha}$. Therefore, the reward with fee $t_{\alpha,1-\alpha}(t)$ is

$$\alpha(1 + e^{(1-\alpha)t/\alpha} - 1)e^{-(1-\alpha)t/\alpha} = \alpha e^{(1-\alpha)t/\alpha} e^{-(1-\alpha)t/\alpha} = \alpha$$

which is equal to Alice's expected reward when she does compute the transaction.

4.3. Allocating computational resources

So far, we have assumed that a miner either begins mining immediately and does not compute the transaction, or computes the transaction in its entirety and then begins mining. We call this single-tasking, as discussed in section 1.3. A natural question is whether single-tasking is the optimal strategy for a miner, or whether they could benefit from dedicating some portion of their compute power to computing the transaction, while simultaneously using the rest of their compute power to mine.

We consider the simplified scenario (still in the proportional time model) where Alice is a miner with hashrate α, and all other miners mine empty blocks. Alice can choose to dedicate some fraction κ of her hashrate to computing a transaction and the rest of her hashrate $(\alpha - \kappa\alpha)$ to mining. Suppose there is a transaction in the mempool that has reward g and takes time $\frac{t}{\kappa\alpha}$ for Alice to compute if she dedicates a κ fraction of her power to the transaction. Once the transaction has been computed, Alice redirects all her power to mining a block including the transaction.

Alice's reward from the next block is 1 if she mines the first block before she has computed the transaction. Her effective hashrate during this period is $\alpha - \kappa\alpha$, and including her, the total hashrate of miners mining during this period is $1 - \kappa\alpha$. Recall that it takes Alice time $\frac{t}{\kappa\alpha}$ to compute the transaction. Therefore, the probability that a block is mined before Alice has computed the transaction is $1 - e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}$. The probability that this block was mined by Alice is her hashrate over the total hashrate, which is $\frac{\alpha-\kappa\alpha}{1-\kappa\alpha}$. Therefore, Alice's expected reward from the period of time where she is computing the transaction is

$$\frac{\alpha - \kappa\alpha}{1 - \kappa\alpha}\left(1 - e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}\right)$$

After Alice has finished computing the transaction, her reward if she mines the next block is $1 + g$. This happens exactly when no block is mined before Alice finishes computing, and the first block mined belongs to Alice. The probability that no block is mined before Alice finishes computing is $e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}$. The probability that Alice mines the first block after she is done computing is α, since all miners are mining at this point. Therefore, Alice's expected reward from the period of time after she has finished computing is

$$\alpha(1+g)e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}$$

Putting these two expressions together, Alice's expected reward is

$$r(\kappa) = \frac{\alpha - \kappa\alpha}{1-\kappa\alpha}\left(1 - e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}\right) + \alpha(1+g)e^{-(1-\kappa\alpha)\frac{t}{\kappa\alpha}}$$

The value of $\kappa \in [0,1]$ that maximizes $r(\kappa)$ is the fraction of mining power that Alice should spend computing the transaction in order to maximize her expected reward.

Below, we examine Alice's rewards $r(\kappa)$ when the transaction fee g is set to exactly the minimum fee required for a miner with hashrate α to work on the transaction: $g = e^{t/\alpha - t} - 1$. We consider a large transaction with $t = 0.5$, meaning it takes a miner with hashrate α $\frac{0.5}{\alpha}$ time units to compute the transaction (where a time unit is the average length of time between blocks). We plot $r(\kappa)$ for various values of α in Figure 1.

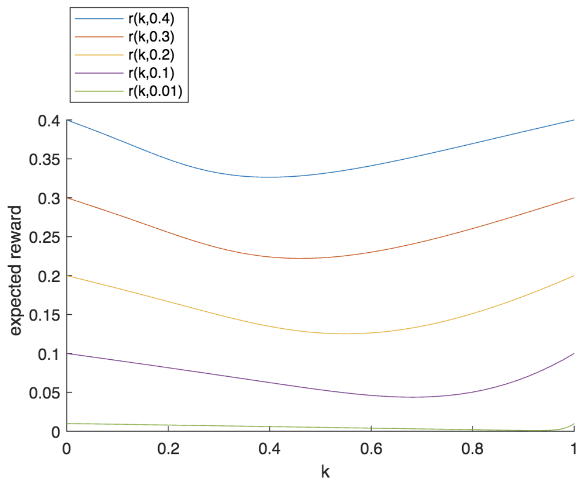


Figure 1: The expected reward for a miner with hashrate $\alpha \in \{0.01, 0.1, 0.2, 0.3, 0.4\}$, compared to the fraction κ of their hashrate dedicated to computing the transaction

Notice that in this example, Alice's reward is always maximized for $\kappa = 0$ or $\kappa = 1$. We suspect this is the case for reasonable values of t and α. This suggests that Alice should either use all of her power to compute the transaction or use all of her power to mine, and our assumption in our previous analysis that she behaves this way is reasonable.

5. A general expression for gas prices

Both the uniform time and the proportional time models have shortcomings in certain scenarios, with the former failing to account for the positive correlations between hashrate and overall network access and computational power, and the latter perhaps assuming a larger amount of correlation than what holds in practice. In this section, we look at a more general setting that encompasses both of the previous models. In particular, we again consider a mempool containing a single contract, but we no longer assume anything about the running time of that contract for different miners (other than the fact that its value for each miner is publicly known to all miners in advance).

Let Alice be a miner with hash rate α, and let $p(x)$ denote the fraction of the hashrate (excluding Alice) mining at time x. Note that p is a (weakly) increasing function, since the total mining hashrate changes only when a miner finishes computing the contract and joins the mining pool. Consider a transaction in the mempool that will take Alice t time to run, and in that case rewards her with g in gas fees. Also let $F(x) = \int_{y=0}^{x}(\alpha + p(y))dy$ denote the cumulative mining hashrate prior to time x if Alice mines from the beginning. Finally, let $f(x)$ denote the mining hashrate at time x, so if Alice mines from the beginning, we would have $f(x) = \alpha + p(x)$ for all $x \geq 0$, and if Alice computes the contract first, we would have

$$f(x) = \begin{cases} p(x) & x \leq t \\ \alpha + p(x) & x > t \end{cases}$$

Note that $f(x)$ is not necessarily the derivative of $F(x)$ (this is only the case if Alice mines from the beginning). We have chosen this (arguably confusing) notation to simplify the final gas price expression in Claim 8.

We start by showing the following claim, which will be useful later.

Claim 7. The probability that no block is mined before time x is given by $e^{-\int_0^x f(y)dy}$ for all $x \geq 0$.

Proof. Let the *survival function* $S(x)$ denote the probability that no block is mined before time x for $0 < x \neq t$. We claim S satisfies the differential equation $S'(x) = -f(x)S(x)$. This is because

$$S'(x) = \lim_{dx \to 0} \frac{S(x+dx) - S(x)}{dx} = \lim_{dx \to 0} -\frac{S(x) - S(x+dx)}{dx}$$

where the numerator denotes the probability of the first ever block arriving in the interval $[x, x+dx]$. This event occurs iff no block arrives before x (which happens with probability $S(x)$), and some block arrives during $[x, x+dx]$ conditioned on surviving up to x. The conditioning can be dropped due to the memoryless property of the proof of work computation, so the probability of the latter event is simply $f(x)dx$. It is easy to confirm that the solution to this differential equation is indeed $S(x) = e^{-\int_0^x f(y)dy}$. □

We are now ready to derive an expression for gas prices as a function of running time.

Claim 8. In this more general setting, Alice needs to be paid

$$g_{\alpha,p}(t) = \frac{1}{e^{\alpha t}} \left(\frac{\int_0^\infty e^{-F(x)}dx}{\int_t^\infty e^{-F(x)}dx} \right) - 1$$

for including the contract in her block in order for the contract to be worth her time to compute.

Proof. We first compute Alice's expected block reward if she does not include the contract and starts mining immediately. We do so by computing Alice's expected block reward during the time interval $[x, x+dx]$ and integrating. We can decompose Alice's expected block reward during $[x, x+dx]$ as the product of probabilities of the following events (since block rewards are normalized to be 1):

- No block is mined before x. By Claim 7, this happens with probability

$$\exp\left(-\int_{y=0}^{x}(\alpha+p(y))dy\right) = e^{-F(x)}.$$

- Someone mines a block during $[x, x+dx]$ conditioned on the previous bullet. By the memoryless property of hashing, the conditioning can be dropped. For infinitesimal dx, the probability that a block is mined during the interval is $(\alpha + p(x))dx$.

- Alice mines a block during $[x, x+dx]$ conditioned on the previous two bullets. The probability of this event is the ratio of Alice's hashrate to the hashrate in that interval and is thus given by $\alpha/(\alpha+p(x))$.

Therefore, Alice's total expected reward from not including the contract is

$$\int_0^\infty \frac{\alpha}{\alpha+p(x)}\cdot(\alpha+p(x))\cdot e^{-F(x)}dx = \alpha\int_0^\infty e^{-F(x)}dx.$$

Next, we will compute Alice's expected (block and gas fee) reward from including the contract if it pays a total fee of g. If Alice successfully computes the transaction and finds a block before anyone else finds a block, her reward is 1 unit of block reward in addition to g units of gas reward. We will again need to compute the probability that Alice succeeds during $[x, x+dx]$. Note that when $x < t$, the probability of Alice succeeding in that interval is zero, since by assumption miners are single-tasking and do not mine while running a contract. For $x \geq t$, Alice's success probability during $[x, x+dx]$ is the product of probabilities of the following events:

- No block is mined before x. By Claim 7, this happens with probability

$$\exp\left(-\int_0^t p(y)dy - \int_t^x(\alpha+p(y))dy\right) = \exp\left(-\int_0^t(\alpha+p(y))dy + \int_0^t \alpha dy\right) = e^{\alpha t - F(x)}.$$

- Alice mines a block during $[x, x+dx]$ conditioned on the previous bullet. Since the proof of work computation is memoryless, the probability of this event is identical to the second and third bullet in the previous list and thus given by α.

Therefore, Alice's total expected reward from including the contract is

$$(1+g)\alpha\int_t^\infty e^{\alpha t-F(x)}dx.$$

We can solve for the minimum gas fee g required to convince Alice to include a contract that takes her t time to run by making her indifferent between including and not including it:

$$g(t) = \frac{\int_0^\infty e^{-F(x)}dx}{\int_t^\infty e^{\alpha t - F(x)}dx} - 1 = \frac{1}{e^{\alpha t}} \left(\frac{\int_0^\infty e^{-F(x)}dx}{\int_t^\infty e^{-F(x)}dx} \right) - 1,$$

which completes the proof. □

As an example, in the case of uniform time, we have that $\alpha + p(y) = \alpha + p$ for $y \leq t$ and 1 for $y \geq t$. Thus, we have

$$F(x) = \begin{cases} (\alpha + p)x & \text{if } x \leq t \\ (\alpha + p)t + (x - t) & \text{if } x > t \end{cases}.$$

Thus, we have

$$\begin{aligned} g(t) &= \frac{1}{e^{\alpha t}} \cdot \frac{\int_0^t e^{-(\alpha+p)x}dx + \int_t^\infty e^{-(\alpha+p)t-(x-t)}dx}{\int_t^\infty e^{-(\alpha+p)t-(x-t)}dx} - 1 \\ &= \frac{1}{e^{\alpha t}} \cdot \frac{\frac{1}{\alpha+p}\left(1 - e^{-(\alpha+p)t}\right) + e^{-(\alpha+p)t}}{e^{-(\alpha+p)t}} - 1, \end{aligned}$$

matching the result in Claim 1. Results in the proportional time model (which we explored in Section 4), including an expression for $g(t)$ for any particular profile of miner hashrates, can be derived similarly from Claim 8.

6. Discussion

The throughput-liveness trade off Block gas limit give an upper bound on both maximum and average throughput of the chain in an explicit way. As demand for gas rises, one might naively suggest increasing block the gas limits can suffice. However, as our analysis shows, there is a fundamental limit to how much raising the gas limit can help: A block whose transaction content takes much longer than the average block time to execute can likely need astronomical gas fees to ever appear on the chain under reasonable assumptions. That is, the a average block time, a low value for which is desired to ensure liveness, in turn poses a bottleneck on throughput in the current state of Ethereum.

Alternative design: allowing more expressive bids We have explored several ways in which gas fees for large transactions transactions might need to grow superlinearly and vary from miner to miner depending on the particular distribution of hashrate among miners. The current mechanism for determining gas fees only allows for submitting a single number to designate the author's willingness to pay per unit of gas. The EIP-1559 proposal for modifying the fee mechanism, which focuses on simplifying the user experience, also utilizes a per-gas type of bid. Our results show that the decision faced by both miners and transaction authors can be very complicated in the case of large transactions, as the authors have to compress their valuation into a single number, and the miners hope to use this one number to interpolate the complicated gas fee expressions. A possible alternative design would allow authors to specify their bids as functions that depend on the amount of gas used by their transaction. Users can thus submit expressions similar to the ones we derived in this paper, allowing them to communicate their valuation better and increasing the likelihood of miners including their transaction. However, while this modification can help slightly in the case of large but not too large transactions (such as ones that take a few seconds to compute but not much larger than the average block time of 10-15 seconds), it cannot overcome the fundamental limits on throughput discussed above.

Alternative design: signaling state-independence We propose a modification that that allows for periodic inclusion of very large transactions in Ethereum blocks. This proposal is motivated by the observation that in our analysis we relied crucially on the assumption that contracts are completely state-dependent and all work goes to waste as soon as a new block is published. Not all tasks are this highly sensitivity to state. For example, it might be desired to outsource a difficult computational task that references nothing about the blockchain to Ethereum miners, who often have very large computational resources at their disposal, either due to lack of access to such resources by the transaction author, or to allow for pointing others to the transaction on the blockchain as *proof* that it has the claimed output.

Suppose a transaction author can somehow guarantee that the miner's work will not be wasted in this way. Suppose furthermore that all miners run all transactions that are every published on the chain. In this case, there gas fees are no longer non-linear as mining instead of computing the transaction now only delays the computation until later rather than completely get rid of it.

More concretely, suppose there is a special "benign" tag that an author can post on a transaction she submits, certifying that the transaction involves no reference to the content of the blockchain. If the miner detects such a reference at any point in the execution, the miner can include the transaction in a block and claim gas fee equal to the number of operations it took before detecting state-dependence. Otherwise, the miner must execute the transaction fully and correctly for his block to be considered valid. The block gas limit rule is also modified so that "benign" transactions do not count toward the gas limit of a block, allowing for arbitrarily large "benign" transactions to be posted.

One possible objection to the above proposal is that it increases the average throughput of the network, introducing additional burden on the full nodes and promoting centralization. In response, note that the extent to which this modification can increase the average throughput of the network can be controlled by having a separate block gas limit for "benign" transactions, which ensures that the total gas consumed on such transactions over n consecutive blocks is capped. The idea is to allow for large increases in max throughput while keeping average throughput relatively small. Each full node can thus allocate some fraction of its compute to benign tasks, and vary this amount depending on current demand for regular (non-benign) transactions by temporarily pausing the execution of benign tasks.

Future directions

- In our analysis, we assumed that miners were myopic: that they did not care about their reward beyond the next block. It would be interesting to see whether the main conclusions of our analysis change if we consider non-myopic miners.
- With EIP-1559, Ethereum will change substantially. It would be interesting to see in more detail the extent to which our analysis carries over to this new setting.
- As discussed above, our work is closely related to the verifier's dilemma. A greater understanding of non-miner incentives with respect to verifying transactions would be helpful.
- We have assumed that miners know exactly how long a contract will take them to compute, and exactly how other miners will behave. Our analysis could be redone with these assumptions relaxed.
- It is possible that there are attacks on the Ethereum network whereby a user creates a contract that is difficult to compute in order to take certain miners offline. For instance, perhaps a small miner could do this in order to gain an advantage over large miners (who will choose to compute the transaction). It may be worth carefully analyzing such attacks, as this could be a potential weak spot in the security of Ethereum in the future.