# Implications of Ether Burn for Computation

Aeron Buchanan, Version 2.1

## Intro

It is important to penalize the computation that contracts perform to keep the work miners have to do worthwhile. In the extreme case, a zero computation cost would allow for infinite loops and Ethereum would hit an impassable barrier. Even the threat of infinite loops would likely result in no one willing to mine. Arbitrary limits on computation are not ideal.

The question is what to do with the payment for computation. Following from Joseph's doc, I add here some thoughts about the implications of simply discarding the payments. Note that much of what I say here is what Joseph and others have already said, but reworded to help (me and) the issues to be considered from other perspectives.

Firstly, a note about the original option, i.e. no burn, where the payments for computation are somehow passed back to other Ethereum wallets (e.g. via miners, or simply spread evenly accross all wallets). In this scenario, there is an (arbitrary) upper bound on computation (beyond the free quota) set as the total amount of Ether divided by the basefee. Of course, the actual practical limit for computation is lower than this because not all Ether is available for computation at a given time and Ether is unlikely to be distributed optimally across contracts to maximize computation. There are also the issues of contract execution order and contract re-funding which have implications on the upper bound (possibly in either direction). The theoretical upper bound, along with the "computation value" of Ether (in "operations per Ether"), increase as the total amount of Ether increases.

Secondly, a note on computational demand. This notion has two sides to it: there is the amount of computation that people would like to do in Ethereum in some pure unbounded abstract way; and there is the amount they are willing to pay for, given the current computational value of Ether. In my discussion here, I try to blur the two (for convenience of analysis) by considering the trend in computational demand for some time period.

Back to Ether burn: with a fixed basefee, the main implication is that computation would then decrease the total amount of Ether. Therefore, burning the Ether for computation leads to one of three outcomes:

1. The Ether supply runs out
2. Computation, beyond the free quota, stops
3. Some interesting equilibrium between the two

Oil might be a good analogy: as the oil supply runs low the amount people are willing to burn reduces. Eventually, we'll get to a point where there is some oil left, but to burn it costs more than any one person is willing to pay. The last scenario is effectively just a reduced upper bound for computation. I would suggest none of these scenarios are desirable.

Therefore, we consider varying the basefee. In order to avoid the above scenarios, the Ether cost of computation has to decrease with increased levels of computation demand. This in itself is interesting as, on first view, this is the opposite of conventional supply-and-demand economics. However, computation supply is, as far as Ether is concerned, infinite, hence the reason to charge for it in the first place. An interesting side-effect is that, by burning Ether, and thus reducing the supply, the computational value of Ether could well be increased.

## Conclusion

I've looked at various approaches to how varying the basefee could be handled and my conclusion is that changing the basefee is probably not a good idea. At best (and I don't know how to achieve this), we just have the non-burn system, but with a reduced upper bound on computation. In general, we have systems that fall into the bad scenarios listed above, but faster.

I think it is also important that we maximize the chances for people to be able to predict things, such as the amount of Ether and the cost of computation. If we make it harder for people to predict these, then we are adding to the instability of the Ethereum platform. Burning computation spend makes predicting the future total of Ether more difficult; changing the basefee makes it even more difficult for people to predict what their Ether will be worth in the future, both in terms of computation and for any external value they may attribute to it. We want it to be easy to predict how much computation your Ether will buy, otherwise we will end up with either too much set aside for contracts or they will be moments when large swathes of contracts run out of funds and stop. If we also want Ether to be taken on as a currency, then instability is undesirable, and better avoided without adding sources of variability.

We might hope that computational demand in Ethereum will vary from block to block in easy-to-predict small amounts (which even then would not mitigate the above concerns to a significant degree - see below), but that is in no way guaranteed. This is a big problem for varying the cost of computation each time-step. Even if everyone is well behaved and contracts are bug-free, it seems reasonable to expect that computational demand within Ethereum will increase exponentially with time, and likely above Moore's law as more people become aware of it and join in. In fact, I would say that this is a desirable target to hope for. As such we must consider the scenario where the long-term growth in computational demand out-paces growth in total Ether because it is pretty much certain to happen. Furthermore, even if we employ a model that has beautiful long-term behaviour, jumps and spikes in demand, particularly during the vulnerable early stages, could result in effectively arbitrary behaviour (as seen on the stock markets for example, except without a fail-safe shut-off switch). This consideration restricts our options: we can in no way guarantee that a variable basefee won't change dramatically without taking on the risk of running out of Ether for computation for some period of time, which is a major cost for Ethereum as an operating system.

Nevertheless, it is far from clear whether there is a better alternative, and so further thought is warranted. There are some aspects that can be considered without resorting to in-depth maths or modelling. The first is what classes of basefee update are viable. The main point here is that the basefee can only be set for a particular block using data from previous blocks, because contracts are necessarily charged as they are executed; the halting problem means that retroactive application of basefee updates is not possible. This means that we have to use a prediction for the amount of computation that will happen in the next block to set a safe (not too high) and sensible (not too low) basefee for that block. Trying to predict more than one block ahead quickly gets dangerous, particularly in the early stages of system progression, and, as mentioned above, having Ethereum collapse in the first year is rather costly. Fortunately, control theory can give us a good handle on modelling feedback systems, so we can turn to a wealth of expertise for the details. Before that though, it must be decided what sort of control to apply, and by this I don't mean which model to use for the change in computational demand, but rather what outcome are we aiming for? Ultimately, any model we decide to implement will occasionally fail as people are irrational: we just hope that Ethereum gets big enough fast enough that craziness averages out, but that is not at all guaranteed. However, we still need to aim for a particular outcome, such as

i) constant total Ether burn for computation

ii) constant value reward for miners

iii) constant percentage Ether burn in relation to total Ether

iv) constant Ether value in terms of computation

v) constant computation

vi) some higher-order target

which increase in complexity down the list. Here is an overview of the basic analysis, if we accept that there will be an exponential demand for computation over time:

i) *constant total burn*: the long-term implication of the this option rules it out immediately: the basefee will reduce to zero or total Ether will reduce to zero.

ii) *constant value reward for miners*: this is a special case of (i) and just results in the basefee reducing to zero and total Ether remaining constant.

iii) *constant percentage burn*: again, the long term behaviour is undesirable: the basefee will reduce to zero and the total Ether will reduce to the amount that is added each year $0.4P$.

iv) *constant computational value*: this is impossible to maintain in the long run and so must also be ruled out.

v) *constant computation*: we could use a varying basefee to set a fixed quota for computation. NB: there is always an implicit ceiling for amount of computation (beyond the free quota) that can be performed at a given time-step, set by the total Ether divided by the basefee, so this option doesn't really solve anything.

vi) *something else*: yet to be considered

A more thorough analysis is in the Maths section below.

At the moment, I still think that transferring the cost of computation to miners is simpler and therefore better.

There are other options: for example varying the free computational quota, which has the advantage of affecting the total amount of Ether less, but they all probably mean people will find it hard to know the future cost of computation.

Beyond the considerations in the introduction, a major long-term problem is the ratio of computational demand to the constant rate at which Ether is added. Inspired by this, we could consider varying the amount of Ether that we add each year based on the amount of computation required for contracts, but, of course, this is just a mechanism for redistributing the cost of computation.

As already mentioned, what follows is a small foray into analysing the various options. I am yet to consider possible emergent behaviour over short and medium terms because so far looking at the long-term properties of our options suggests that all models simply store up problems for Ethereum in the future, if they don't destroy it first.

## Maths

In the following sections, some basefee update strategies are investigated. As discussed above, these strategies are pretty much useless, but I've included my analysis here because it hopefully illuminates my approach so far.

The total cost of computation in a given time-step (block) $n$ is

$$C_n = N_n(k_n - c)\mathbf{x}_n = \ddot{U}_n\mathbf{x}_n \tag{1}$$

where

· $N$ is the number of contracts (variable)

· $c$ is the free computation quota (*currently fixed*)

· $k$ is a measure of average computation demand per contract (variable)

· $\ddot{U}$ is a measure of the total computational demand in time-step $n$

· $\mathbf{x}$ is the basefee (*controlled*)

In my opinion, it is going to be useful to look at the computational value of Ether, i.e. operations per Ether:

$$v_n = \frac{\ddot{U}_n}{T_n} \tag{2}$$

Also, ignoring general loss of Ether through forgotten wallets, etc, we have that the total amount of Ether progresses according to

$$T_n = T_{n-1} + R - \ddot{U}_{n-1}\mathbf{x}_{n-1} \tag{3}$$

which are values that might give us a handle on emergent behaviour, where

· $T_n$ is the total amount of Ether and

· $R$ is the miner's reward in one time-step (*fixed*)

### First Option: fixed total cost

One way of guaranteeing that the total amount of Ether remains stable is to (approximately) fix the total cost of computation for any given time-step:

$$\mathbf{x}_n = \frac{X_0}{\ddot{U}_n^*} \tag{4}$$

where $^*$ has been used to signify a prediction. Note that this means that the analysis here is best-case. We immediately see that as computational demand $\ddot{U}$ increases unbounded, the basefee will reduce to an arbitrarily small value. If the growth of computational demand is exponential, the basefee will reduce to zero quite quickly after a certain point (that point being defined by the growth factor and $X_0$). It should be emphasised that there is nothing to stop computation increasing unbounded in this scheme.

Ignoring that we should definitely not use this approach, let's continue to look at possible long-term trends. Firstly, the recurrence relation for $T_n$ gives a closed-form equation for the total

$$T_n = T_0 + n(R - X_0) \tag{5}$$

i.e. if $X_0 > R$ Ether will eventually run out, whatever the computation demand, and if $X_0 < R$, Ether grow will progress linearly, but at a reduced rate. We could, for example, set $X_0 = 0.5R$ and ensure that the total amount of Ether increases at $0.2P$ per year instead of $0.4P$. We can also look at the change in the computational value of Ether

$$\begin{aligned} v_n - v_{n-1} &= (\ddot{U}_n - \ddot{U}_{n-1})\frac{1}{T_{n-1}} + \ddot{U}_{n-1}\frac{X_0 - R}{T_{n-1}^2 + T_{n-1}(R - X_0)} && (6) \\ &\approx (\ddot{U}_n - \ddot{U}_{n-1})\frac{1}{T_{n-1}} \quad \text{while} \quad T_n \gg \ddot{U}_n && (7) \end{aligned}$$

which mainly demonstrates that when people try to estimate the future computational value of their Ether they will be able to do so trivially for a while, but eventually the offset based on the computational demand history will have to be taken into account as demand will eventually overtake the total amount of Ether whatever parameters we pick.

It might be more helpful to consider the proportional change in value:

$$\begin{aligned} \frac{v_n}{v_{n-1}} &= \frac{\ddot{U}_n}{\ddot{U}_{n-1}}\left(\frac{T_{n-1}}{T_{n-1} + R - X_0}\right) && (8) \\ &= \frac{\ddot{U}_n}{\ddot{U}_{n-1}} \quad \text{eventually} && (9) \end{aligned}$$

which does, at least, give a generally good handle on things.

## Special case: constant value for miners

With Ether burn miners simply get a fixed reward, $R$ for mining a block, irrespective of how much computation they had to do to run all the contracts. We might suggest that we use the basefee to adjust the value of their reward. For example, fixing the value of their reward:

$$\frac{R}{T_n} = \text{constant} \qquad (10)$$

which means that

$$\begin{aligned} \frac{R}{T_n} &= \frac{R}{T_n - 1} && (11) \\ &= \frac{R}{T_n + \mathbf{x}_n\ddot{U}_n - R} && (12) \\ \mathbf{x}_n &= \frac{R}{\ddot{U}_n} && (13) \end{aligned}$$

i.e. the only way to achieve this is to burn all the Ether we add each year. Not useful.

## Another Option: fixed proportion

The next option is to fix the proportion of the current total that computation will cost:

$$\mathbf{x}_n = \frac{X_0' T_n^*}{\ddot{U}_n^*} \qquad (14)$$

where $X_0'$ is the parameter that sets the fraction of the total we aim to charge. Repeating the above analysis, we get

$$\begin{aligned}
T_n &= T_{n-1} + R - X_0' T_{n-1} && (15)\\
&= T_{n-1}(1 - X_0') + R && (16)\\
&= (1 - X_0')^n T_0 + R\frac{1 - (1 - X_0')^n}{X_0'} && (17)\\
&= \frac{R}{X_0} \quad \text{eventually, noting } (1 - \mathrm{X}_0') < 1 && (18)
\end{aligned}$$

Trending to a fixed value, whatever the progression of computational demand, is not a desirable attribute. Anyway, let's look at how the value changes:

$$\begin{aligned}
v_n - v_{n-1} &= (\ddot{U}_n - \ddot{U}_{n-1})\frac{1}{T_{n-1}} + \ddot{U}_{n-1}\frac{X_0 - R}{T_{n-1}^2(1 - X_0) + T_{n-1}R} && (19)\\
&\approx (\ddot{U}_n - \ddot{U}_{n-1})\frac{1}{T_{n-1}} \quad \text{while} \quad T_n \gg \ddot{U}_n && (20)
\end{aligned}$$

and

$$\begin{aligned}
\frac{v_n}{v_{n-1}} &= \frac{\ddot{U}_n}{\ddot{U}_{n-1}}\left(\frac{T_{n-1}}{T_{n-1}(1 - X_0') + R}\right) && (21)\\
&= \frac{\ddot{U}_n}{\ddot{U}_{n-1}} \quad \text{eventually, or if} \quad X_0' \ll 1 && (22)
\end{aligned}$$

which are relatively straight forward at least.

## Option C: fixed value

In this scheme we would fix the computational value of Ether, thus:

$$\begin{aligned}
v_n &= \frac{\ddot{U}_n}{T_n} = V_0 && (23)\\
T_n &= \frac{\ddot{U}_n}{V_0} && (24)
\end{aligned}$$

which, on substitution into the Ether total update equation, gives

$$\begin{aligned}
T_n &= T_{n-1} + R - U_{n-1}x_{n-1} && (25)\\
\frac{U_n}{V_0} &= \frac{U_{n-1}}{V_0} + R - \ddot{U}_{n-1}\mathbf{x}_{n-1} && (26)\\
x_n &= \frac{R + \frac{1}{V_0}(\ddot{U}_n - \ddot{U}_{n+1})}{\ddot{U}_n} && (27)\\
&= \frac{R}{U_n} + \frac{1}{v_0}(1 - \mu) \quad \text{for exponential growth} && (28)\\
&\approx \frac{1}{V_0}(1 - \mu) \quad \text{eventually} && (29)
\end{aligned}$$

which, unhelpfully, tends to a fixed negative value (during growth in computational demand at least), requiring us to pay miners for computation: exactly the process we want to avoid.

## A Couple of Update-focussed Models

Joseph's suggested update equation, and when substituting $C_n$ into it, is

$$\begin{aligned}
\mathbf{x}_n &= \mathbf{x}_{n-1} - A(eph_n - eph_{n-1}) && (30)\\
&= \mathbf{x}_{n-1} - A(C_n - C_{n-1}) && (31)\\
&= \mathbf{x}_{n-1}\frac{1 + A\ddot{U}_{n-1}}{1 + A\ddot{U}^*_n} && (32)
\end{aligned}$$

where

- · $\ddot{U}$ is the total demand for computation, beyond the free quota, and
- · $A$ is a magic parameter

which clearly displays the expected results:

$\ddot{U}_n > \ddot{U}_{n-1}$: increasing computational demand decreases $\mathbf{x}$

$\ddot{U}_n < \ddot{U}_{n-1}$: decreasing computational demand increases $\mathbf{x}$

This defines a recurrence relation, so we can simplify:

$$\begin{aligned}
\mathbf{x}_n &= \mathbf{x}_{n-2}\frac{(1 + A\ddot{U}_{n-2})}{(1 + A\ddot{U}_{n-1})}\frac{(1 + A\ddot{U}_{n-1})}{(1 + A\ddot{U}_n)} && (33)\\
&= \mathbf{x}_{n-2}\frac{(1 + A\ddot{U}_{n-2})}{(1 + A\ddot{U}_n)} && (34)\\
&= \mathbf{x}_0\frac{1 + A\ddot{U}_0}{1 + A\ddot{U}_n} && (35)\\
&= \frac{A_0\mathbf{x}_0}{1 + A\ddot{U}_n} && (36)\\
&\approx 0 \quad \text{eventually, as computation grows} && (37)
\end{aligned}$$

because the pre-genesis values can be arbitrarily set to create $A_0$ for convenience.

The conclusion is that, with this update procedure, the current basefee is only actually a function of the current time-step's statistics, as all the rest cancels out and we have that, as computational demand increases, the cost of computation decreases in an exponential way, eventually becoming zero. As demand decreases, the cost increases to some upper limit, $A_0\mathbf{x}_0$.

Again, we can look at the implications for the total amount of Ether. The change in one time-step is

$$T_n - T_{n-1} = R - \frac{A_0\mathbf{x}_0\ddot{U}_{n-1}}{1 + A\ddot{U}_{n-1}} \qquad (38)$$

so we have that there is a tipping point at

$$\ddot{U}_n \lessgtr \frac{R}{A_0x_0 - AR} \qquad (39)$$

below which total Ether will rise and above which (the expected norm) total Ether decreases. In general we can see that if computational demand rises unbounded

$$T_n - T_{n-1} = R - \frac{A_0\mathbf{x}_0}{A} \quad \text{eventually} \tag{40}$$

then the long-term tipping point is

$$\frac{A_0\mathbf{x}_0}{A} \lesseqgtr R \tag{41}$$

that is, we can pick our parameters to eventually achieve the "fixed total cost" model rejected above, but with more potential for instability up to that point.

If we assume that the demand for computation $\ddot{U}$ increases exponentially, that is $\ddot{U}_n = \mu\ddot{U}_{n-1}$, then the tipping point can be expressed in terms of the growth factor

$$\mu \lesseqgtr \sqrt[n]{\frac{R}{\ddot{U}_0(A_0x_0 - AR)}} \tag{42}$$

with which we could tune the point at which things go badly wrong if we wanted.

Continuing for interest, also of note is the behaviour of the second derivative

$$T_n - 2T_{n-1} + T_{n-2} = A_0x_0\frac{\ddot{U}_{n-2} - \ddot{U}_{n-1}}{(1 + A\ddot{U}_{n-2})(1 + A\ddot{U}_{n-1})} \tag{43}$$

$$= A_0x_0\frac{1-\mu}{(1 + A\ddot{U}_{n-2})(1 + A\mu\ddot{U}_{n-2})}\ddot{U}_{n-2} \tag{44}$$

confirming the desire that if $\mu > 1$ the rate of change of total Ether will decrease, but will increase if $\mu < 1$ (i.e. gets closer to zero), and stay constant if computational demand does not change.

As above, we can look at the way that the computational value of Ether, $v_n = \frac{\ddot{U}_n}{T_n}$, changes as computational demand changes:

$$v_n - v_{n-1} = \frac{\ddot{U}_n}{T_n} - \frac{\ddot{U}_{n-1}}{T_{n-1}} \tag{45}$$

$$= \frac{(\ddot{U}_n - \ddot{U}_{n-1})(1 + A\ddot{U}_{n-1})T_{n-1} - \ddot{U}_{n-1}(R - \ddot{U}_{n-1}(AR - A_0x_0))}{T_{n-1}((T_{n-1} + R)(1 + A\ddot{U}_{n-1}) - \ddot{U}_{n-1}A_0x_0} \tag{46}$$

$$= (\ddot{U}_n - \ddot{U}_{n-1})\mathsf{F}(T_{n-1}, \ddot{U}_{n-1}) + \mathsf{G}(T_{n-1}, \ddot{U}_{n-1}) \tag{47}$$

which, even when considered in terms of the change in computation demand, is just crazy.

## Modification

If we don't trust the ability to predict the computational demand in the next time-step, we need to modify the above slightly, so that the current basefee $\mathbf{x}_n$ is based only on previous statistics (i.e. only quantities from time-step $n-1$ or earlier). This is because contracts are necessarily charged as they are run, not afterwards.

As such, we might consider the following update schema

$$\mathbf{x}_n = \mathbf{x}_{n-1} - A(\ddot{U}_{n-1}\mathbf{x}_{n-1} - \ddot{U}_{n-2}\mathbf{x}_{n-2}) \tag{48}$$

$$= \mathbf{x}_0 - A\ddot{U}_{n-1}\mathbf{x}_{n-1} \tag{49}$$

$$= \mathbf{x}_0\left[1 + \sum_{k=1}^{n}\left\{(-A)^k\prod_{i=1}^{k}\ddot{U}_{n-i}\right\}\right] \tag{50}$$

and if computational demand follows a geometric progression, $\ddot{U}_n = \mu \ddot{U}_{n-1}$, we have

$$\mathbf{x}_n = \mathbf{x}_0 \left[ 1 + \sum_{k=1}^{n} (-A\ddot{U}_0 \mu^{n-\frac{1}{2}})^k \mu^{-\frac{k^2}{2}} \right] \tag{51}$$

which I haven't found the tools to investigate analytically, but it's clear enough that computational demand can grow enough to make the absolute size of $\mathbf{x}$ arbitrarily large (positive or negative), pointing out just how easy it would be for things to go horribly wrong.