EPR-ereum
bridging different worlds
WARNING: unfinished draft

Benjamin Bollen

July 27, 2017

Overview

Proof-of-Work mining successfully secures Ethereum, however it does not allow the computational power of Ethereum to scale as more full nodes join the network.

We present an overlay protocol to scale the capacity of decentralised applications on public Ethereum. Our solution manages all value on Ethereum and provides an on-chain marketplace for pooling computational resources of full nodes allowing them to contribute and earn gas rewards linearly from the execution power they provide to Ethereum.

The solution does not modify the scaling properties of Ethereum directly, rather it aims to get more computational *gas mileage* out of the same Ether by minimising the verification burden put on the Proof-of-Work miners. To this end we introduce a Byzantine fault-tolerant, staked meta-consensus mechanism. This mechanism allows us to scale computational capacity with the number of groups of nodes available. By moving the verification burden away from the Proof-of-Work miners we bypass the *verifiers dilemma*[1] while retaining the full replay history of Ethereum.

We detail the protocol and discuss an early analysis of attack vectors. We describe possible use-cases and review the gains by enabling Eprereum for the application. At the end we place this proposal in relation to existing work on scaling Ethereum.

[1]The verifiers dilemma states that the verification burden for honest miners in Nakamoto consensus systems must be small compared to the sealing effort lest the honest miners be vulnerable to attacks. We will discuss the dilemma in more detail further.

1 It's all about gas

In Ethereum gas accounts for every execution step. A transaction includes a *gas price* and a *gas limit*. The gas limit sets the maximum gas that can be burnt during the execution of this transaction. When the execution completes the total gas used is deducted from the account balance of the transaction *sender* at the set gas price[2].

In late spring of 2017 a steep increase in the number of transactions processed on the Ethereum blockchain has been registered. All the while the average gas used per transaction has remained in first approximation constant. The accompanying higher market valuation of Ether had the price per transaction skyrocket. In response the average gas price has started a correction downwards at the time of this writing. [graph]

This demonstrates that the Ethereum protocol has scaled successfully under an increase of the number of transactions. However, it leaves unanswered how Ethereum can scale for more computationally intensive applications. We will argue that it is not only the price that is prohibitive to build applications that consume more gas. It has been highlighted

[2]The gas price is set in Ether per gas [ETH], where gas is a number.

that Nakamoto consensus systems have an inherent requirement to keep the block validation to a mimimum. Consequently this keeps the gas price for transactions validated by the Proof-of-Work miners high, as the total block fee, $f = \sum_{tx} \text{gas} \cdot \text{gasprice}$. From this it is clear that for honest miners security and profitability are optimised with a supply of many low-gas, high-gas price transactions.

The security that Proof-of-Work miners generate is rooted in the adversarial race the miners are in to solve the block sealing puzzle first. Only the miner who contributes the block is awarded the block rewards and block fee. Other miners are expected to validate the correctness of the transactions in a block for the good of the network without reward. They obviously have an incentive to make sure the block does not contain invalid transactions, because that would invalidate all work they build on top of it. Verifying transactions of mined blocks, however, carries the risk of falling behind on mining the new block. This is a brief summary of what has been understood as the verifiers dilemma[3][Teutsch].

If the verifiers dilemma is implied by the Nakamoto consensus and its reward structure, then it can be avoided by examining other incentive structures. The mining paradigm to date forms the foundation of the two biggest permissionless blockchains and as such our objective in this work is not to replace it. Rather we will present an opt-in, overlay protocol that constructs a new scaling dimension for decentralised applications on Ethereum.

2 Bridging different worlds

Going forward we define the results of the Proof-of-Work consensus of Ethereum to be true. We note that a Nakamoto consensus engine can roll back state, however, our construction will be relative to a block and as such local to whichever branch of the consensus engine eventually accumulates most work.

We briefly introduce a notation. A transaction t on Ethereum transforms the full state S to a new state S'. The Ethereum Virtual Machine (EVM) is explicitly constructed to be serial in its execution, so we can compose transactions:

$$\begin{aligned} t_1 \circ t_2 &: S \to S'' \\ \text{where } t_1 &: S \to S', t_2 : S' \to S''. \end{aligned} \tag{1}$$

For a transaction that calls a constant function at the end of its execution we can note:

$$\begin{aligned} t = \tilde{t} \circ \tilde{t}_c &: S \to S', r \\ \text{where } \tilde{t} &: S \to S', \tilde{t}_c : S' \to S', r \end{aligned} \tag{2}$$

with r the result returned from the function call $\tilde{t}_c$.

While the execution of a single transaction is atomic and serialised, smart contract developers need to code smart contracts with the understanding that the order of transactions is not under their control as it is determined by the block formation of Proof-of-Work miners. We therefore introduce an *asynchronous callback* $\cdot$ composition we can use to construct an asynchronously composed transaction:

$$t = \tilde{t}_0 \left(\prod_{i=1}^{N} \circ \, \tilde{t}_i \circ \tilde{t}_{c,i} \right) \circ \tilde{t}_{N+1} \circ \left(\prod_{i=1}^{N} \cdot \, t_{r,i} \right), \tag{3}$$

where $\tilde{t}_{c,i}$ call constant functions and $t_{r,i}$ call (non-constant) functions with the asynchronously returned result r_i. Where we require that $\circ \, (t_{r,1} \cdot t_{r,2})$ must equal $\circ \, (t_{r,2} \cdot t_{r,1})$, as the result will be returned through the Nakamoto consensus algorithm, and no guarantee on the order can be given. Note this is not more difficult than standard asynchronous programming as the program has no control over the scheduler that orders the asynchronous callbacks.

Without loss of generality we will further consider transactions of the form

$$t = \tilde{t} \circ \tilde{t}_c \cdot t_r, \tag{4}$$

where $\tilde{t}$ and t_r write to the Ethereum state, and $\tilde{t}_c$ only reads from the Ethereum state, returning a single result r.

As an overlay protocol on Ethereum, all Epreum nodes are constructed to verify the Ethereum blockchain. As a result, we can use smart contracts

[3] see appendix for more details (todo)

on Ethereum to construct a deterministic, global[4] one-to-many communication channel without incurring an overhead on the number of Eprereum nodes in the network.

Communication in the other direction (from Eprereum nodes back to Ethereum), however, is expensive as it requires transactions to be validated by the Proof-of-Work miners at a high gas price.

With the understanding that the execution of $\tilde{t}_c$ in (3, 4) only requires read-access to the Ethereum state, we construct Eprereum as an overlay protocol to execute those constant function calls off the Ethereum blockchain. In return we have to incur an overhead on Ethereum in a Byzantine fault-tolerant meta-consensus process that resolves whether the results r returned to Ethereum in (replicated) transactions t_r have reached consensus.

For this construction to be worthwhile two assumptions need to be fulfilled. The incentive structure for Eprereum must be such that the verifiers dilemma is circumvented and honest nodes can benefit from high-gas, low gas price transactions. Secondly, the overhead incurred by the meta-consensus process needs to be offset by a lower gas price on Eprereum than on Ethereum.

This second condition is easily quantified. If we call g_0 the gas consumed by $\tilde{t}$ in (4), g_c the gas consumed by $\tilde{t}_c$, and g_m the gas consumed by the meta-consensus for concluding r to be true, then g_0 and g_m are charged at the Ethereum gas price p_ETH. If we would simply execute $\tilde{t}_c$ as a normal function call on Ethereum, we would save the gas cost of the overhead incurred by the meta-consensus, however, if we can execute $\tilde{t}_c$ at an Eprereum gas price $0 < p_\mathrm{EPR} < p_\mathrm{ETH}$, we find that it is cheaper if

$$p_\mathrm{EPR} < p_\mathrm{ETH}\left(1 - \frac{g_m}{g_c}\right). \tag{5}$$

As g_m is determined by the implementation of Eprereum and a lower bound on g_c is known at compile time by the application calling on Eprereum, the decision to outsource $\tilde{t}_c$ to Eprereum can be made dynamically on Ethereum, ensuring that the gas price on Eprereum is effectively capped by (5).

To address the ability for Eprereum nodes to execute high-gas calls, Eprereum nodes are orchestrated by smart contracts on Ethereum. From the perspective of the Eprereum nodes this sets rules of interaction that govern the rewards and deposits they can earn and loose while operating in service of Ethereum requests.

To optimise the signal-to-noise ratio for Eprereum to Ethereum communication, the Eprereum nodes are grouped and the actions of the group are staked by the combined stake of all nodes in the group. Actions of Eprereum groups are given to Ethereum through Interblockchain Communication (IBC) messages designed to optimise the communication cost for the meta-consensus process and the Eprereum network orchestration.

3 Connecting the wires

4 Steering network health

[4]Note that this global symmetry is only valid under the stated assumption that the Eprereum network is invariant under the probabilistic finalisation of Ethereum. Here without loss of generality we assume that the global symmetry is valid, but this is a requirement for an implementation to observe.