EPR-ereum
bridging different worlds
WARNING: unfinished draft v0.1.16
all content subject to change and corrections

Benjamin Bollen

August 5, 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; but we do not yet present implementation details. This document is intended to convey a proposal and invite critique and corrections. We 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 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]

[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.

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

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 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:

$$t_1 \circ t_2 : S \to S'' \qquad (1)$$
$$\text{where } t_1 : S \to S', t_2 : S' \to S''.$$

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

$$t = \tilde{t} \circ \tilde{t}_c : S \to S', r \qquad (2)$$
$$\text{where } \tilde{t} : S \to S', \tilde{t}_c : S' \to S', r$$

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:

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

where $\tilde{t}_{c,i}$ call constant functions and $t_{r,i}$ call (non-constant) functions with the asynchronously returned result r_i. 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

$$t = \tilde{t} \circ \tilde{t}_c \cdot t_r, \qquad (4)$$

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.

[3] see appendix for more details (todo)

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

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

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).

2.1 Setting the rules

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

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 increase 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.

An Eprereum group itself is organised as a Tendermint Proof-of-Stake network that has its voting power reflected on Ethereum, where Eprereum tokens are equally bonded. An Eprereum group itself though does not accept Ethereum call transactions. It has specific transaction types to follow the Ethereum blocks as they are mined by Ethereum nodes and to self-organise the execution of calls $\tilde{t}_c$ as they occur on Ethereum. Eprereum nodes finally orchestrate to send interblockchain transactions back to the Ethereum to report on execution results and network staking.

2.2 Accepting results

Let an Eprereum group G^α be composed of Eprereum nodes g_i^α each represented by a public key. We construct a `CommitResult` transaction to contain a `Commit` as the hash of two merkleroots: first, the merkleroot of a set of results to be returned, where

[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.

the blockhash of the Ethereum block associated with these results is the first leaf; second, the merkleroot of the set of signatures of the results root signed by the Eprereum nodes g_i^α in group G^α.

A given block in the Ethereum blockchain can introduce a new ordered set of calculation requests when smart contracts call on the Eprereum scheduling contract. These jobs $\tilde{t}_{c,m}$ with $m = 1 \dots N$ have accompanying results r_m. However, if our objective is to outsource the calculation of r_m to the Eprereum groups G^α, then we want to not calculate $\tilde{t}_{c,m}$ on Ethereum (even though possible - see (5) for the sensibility-inequality) and as such r_m is unknown.

To accept a result r_m back into Ethereum we require that all groups $G^{\{\alpha\}_m}$ commit unanimously to the identical result for r_m. Here we have noted $G^{\{\alpha\}_m}$ as the subset of all groups G^α that are required to return a result for $\tilde{t}_{c,m}$. We will later detail how we can tune the subset selection rule. For now, we can simply consider some groups returning the result.

A group G^α can calculate for an ordered set of job requests $\tilde{t}_{c,m}$ in a given Ethereum block the corresponding results r_m^α. These results combined with the blockhash of the Ethereum block they refer to provides a merkleroot that all Eprereum nodes in the group can sign. These signatures can be stored in a second merkletree, the root of which can be hashed together with the results root to obtain the group commit C^α.

The group commit C^α for a set of job requests $\tilde{t}_{c,m}$ provides a proof point to the Ethereum blockhash, the results r_m^α the group claims to be true, and the signatures of the Eprereum nodes g_i^α in this group G^α signing off on these results.

The group that has done the work to calculate r_m now needs to make this claim to the Ethereum network, but without revealing the results, as that would allow other groups to copy the results and submit them without doing the work, and claiming the rewards nonetheless. Hence in order to ensure that claims by different groups G^1, G^2 have been calculated independently, the groups do not yet share their commit C^1, C^2 but use it encrypt the results r_m^1, r_m^2 as follows.

First note that any node g_i^α can use the commit C^α as a proof to speak on behalf of the group G^α, as it proofs their signatures. To encrypt the results any node in the group can generate privately a secret pseudorandom bitstring S that is a concatenation of bitstrings s_m for every result r_m, where the length of s_m is minimal to guarantee that no two s_m repeat. The node then submits to Ethereum the encrypted results e_m^α:

$$e_m^\alpha = r_m^\alpha \oplus H(s_m, C^\alpha) \tag{6}$$

It suffices for a single node member g_i^α of the group to commit the encrypted results on behalf of the whole group G^α, because if any other group member g_j^α would later refute the results, the committing node can present the merkle proof of the refuting node's signature to the root C^α; failure to present the merkle proof is sufficient cause to take the deposits of the committing node. As a result the combined deposits of all the group members is used to stake in favour of the results e_m^α.

With all encrypted results $e_m^{\{\alpha\}_m}$ received and staked on Ethereum for all groups $G^{\{\alpha\}_m}$ required to commit results, the first phase is closed. Note that as the Eprereum groups can reach consensus much faster than Ethereum, in the ideal case all encrypted results are included in the next (two) Ethereum blocks mined following the block requesting the results.

When Eprereum nodes observe a sufficient staking for the first commit phase, they can initiate the second phase. In order to reveal the results e_m^α that are saved on Ethereum the committing node g_i^α needs only to submit the group commit C^α and the node secret S for Ethereum to calculate the results r_m^α claimed by the group:

$$r_m^\alpha = e_m^\alpha \oplus H(s_m, C^\alpha) \tag{7}$$

For any m the result r_m is accepted if all results revealed $r_m^{\{\alpha\}_m}$ for all $\{\alpha\}_m$ are identical.

2.3 Divide and conquer

3 Connecting the wires

4 Steering network health