DSRV©

EIP-4844: Proto-danksharding

in-house protocol study, 4PM Roundtable

Jeongho Jeon & Youngbin Park

2024-06-05

DSRV (All That Node, WELLDONE Studio)

Data Availability

- Availability – can I use it when I want?
  for example, “Open 10-9“, 99.99%
- Can I read chain data when I want?
- Plasma L2 (commit-chains) post state hashes on L1 periodically.
  If malicious L2 operators conceal L2 transaction details, public cannot prove L2 fraud. Plasma lacks data availability.
  see also Vitalik’s talk at Ethereum Meetup (2017-09-20, YouTube)
- “state” means token balance, NFT ownership, and swap pool details, and so one.
- cf. Data Retrievability = Data Availability + historical archive

Rollup

Rollup L2s write extra data (transactions or both changed state and zero knowledge proofs) for public verification in Ethereum L1.

Rollups are less scalable than Plasma due to extra data, but offer data availability.

A rollup-centric ethereum roadmap

ethereum-roadmap, layer-2

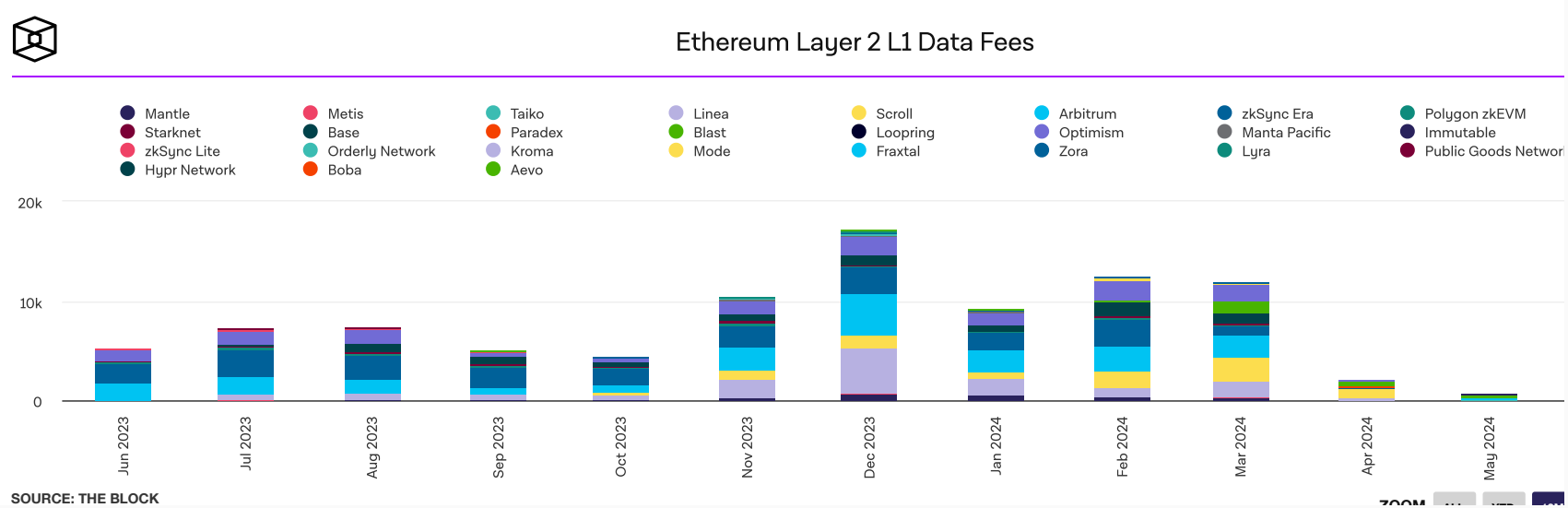
vbuterin 4 Oct 2020

What would a rollup-centric ethereum roadmap look like?

Last week the Optimism team announced 755 the launch of the first stage of their testnet, and the roadmap to mainnet. They are not the only ones; Fuel 479 is moving toward a testnet and Arbitrum 329 has one. In the land of ZK rollups, Loopring 364, Zksync 365 and the Starkware-tech-based Deversifi 279 are already live and have users on mainnet. With OMG network's mainnet beta 319, plasma is moving forward too. Meanwhile, gas prices on eth1 are climbing to new highs, to the point where some non-financial dapps are being forced to shut down 1.0k and others 301 are running on testnets.

The eth2 roadmap offers scalability, and the earlier phases of eth2 are approaching quickly, but base-layer scalability for applications is only coming as the last major phase of eth2, which is still years away. In a further twist of irony, eth2's usability as a data availability layer for rollups comes in phase 1, long before eth2 becomes usable for "traditional" layer-1

Expensive for L2s



see also L2Fees.info

Dencun hardfork
on March 13, 2024

Blob

- 128 kbytes size
- up to 6 blobs in a block (maximum 2 Tb per year)
- separate gas system for blobs
  blob base fee goes up when more than 3 blobs in a block,
  blob base fee goes down when less than 3 blobs in a block.
- keep up to 4096 epochs (about 18 days, maximum 100 Gb)
  It is enough time to challenge and correct malicious L2.
  cf. EIP-4444 (History Expiry, chain data for 1 year only) is under discuss.

Ethereum storage types

The lower the cheaper. (Below gas prices are simplified.)

STORAGE (625 gas/byte × 10 gwei) "state" of smart contacts
All nodes store STORAGE forever.
Smart contracts can read and write STORAGE.

CALLDATA (16 gas/byte × 10 gwei) input data for smart contacts
L2s have used CALLDATA in the place of BLOB before dencun hardfork.
Smart contracts can read CALLDATA.
Vitalik proposed EIP-7706: Separate gas type for calldata last month.

LOG (8 gas/byte × 10 gwei) events during smart contract execution
Smart contacts write LOG, but cannot read.

TRANSIENT STORAGE (3 gas/byte × 10 gwei, EIP-1153, new in dencun)
Smart contracts can read and write TRANSIENT STORAGE,
but discard it after transaction execution.
cf. MEMORY (0.1 gas/byte × 10 gwei)

BLOB (1 gas/byte × 1 wei, EIP-4844, new in dencun)
Nodes store BLOG for a while.
Smart contacts cannot read and write BLOG.

Comparison to storage blockchains

Storage blockchains focus on incentive, and prevent abusers that seek rewards while not providing storage. They use storage dedicated consensus like PoST (Proof of Spacetime). Complex computation may delay data retrieval. When the network has a few copies of data, it recovers copies.

EIP-4844 deals with smaller data than storage blockchains, and serves chain data without delay. It doesn't keep persistent data. see also fisherman's dilemma.

In the future

- Users (i.e. L2s) and EL are not necessary of more major updates in ~~Proto-~~Danksharding.
- will increase the number of blobs (32, 64, 128, 256 or 4096) while monitoring Ethereum network (EIP-7659 or EIP-7691)
  see also Youngbin's blob propagation latency chart
- may switch from KZG to something else (e.g. Merkle trees + STARK)
- DAS (Data Availability Sampling) – all nodes do not need to store all blobs. nodes store a part of blobs, and a client can restore full blobs.
- Celestia, Avail, EigenLayer EigenDA, Starkware DAC (Data Availability Committee)/Validium/Volition, zkPorter, NEAR, ...
- Modular Blockchain – consensus, execution, settlement, and data availability

Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam[1], Alberto Sonnino[1], and Vitalik Buterin[2]

[1] University College London
{m.albassam,a.sonnino}@cs.ucl.ac.uk
[2] Ethereum Research
vitalik@ethereum.org

Abstract. Light clients, also known as Simple Payment Verification (SPV) clients, are nodes which only download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid. Typically, instead of validating block data, they assume that the chain favoured by the blockchain's consensus algorithm only contains valid blocks, and that the majority of block producers are honest. By allowing such clients to receive fraud proofs generated by fully validating nodes that show that a block violates the protocol rules, and combining this with probabilistic sampling techniques to verify that all of the data in a block actually is available to be downloaded, we can eliminate the honest-majority assumption for block validity, and instead make much weaker assumptions about a minimum number of honest nodes that rebroadcast data. Fraud and data availability proofs are key to enabling on-chain scaling of blockchains (*e.g.,* via sharding or bigger blocks) while maintaining a strong assurance that on-chain data is available and valid. We present, implement, and evaluate a novel fraud and data availability proof system.

1 Introduction and Motivation

As cryptocurrencies and smart contract platforms have gained wider adoption, the scalability limitations of existing blockchains have been observed in practice. Popular services have stopped accepting Bitcoin [26] payments due to transac-

Bitcoin Cash: a short-term data availability layer for ethereum?

Sharding

vbuterin 1 Jul '19

Conceptual background: Layer 2 state schemes 195

Summary of conceptual background: there's a large space of quite powerful and effective scalability solutions that rely on a non-scalable *computation layer* (ie. the current ethereum chain suffices) plus a scalable data layer. The general principle of these techniques is that they use interactive computation techniques (eg. Truebit 82) to compute the state on the ethereum side, crucially relying on the data availability verification guaranteed by the data layer to make sure that fraudulent submissions can be detected and heavily penalized. One could use techniques like this to make a highly scalable general-purpose EVM-like system.

In the longer term (1+ year out) the scalable data layer is going to be ethereum 2.0, because its planned 10 MB/sec data throughput is much higher than that of any existing blockchain. In the shorter term, however, we can start working on these techniques *immediately* by using existing blockchains, particularly those that have lower transaction fees per byte than ethereum, as the data layer. Bitcoin Cash arguably fits the bill perfectly for a few reasons:

- High data throughput (32 MB per 600 sec = 53333 bytes per sec, compared to ethereum ~8kb per sec which is already being used by applications)
- Very low fees (whereas BTC would be prohibitively expensive)
- We already have all the machinery we need to verify Bitcoin Cash blocks inside of ethereum thanks to http://btcrelay.org/ 146 ; we just need to repoint it to the BCH chain and turn it back on. Verifying BCH blocks is also quite cheap compared to eg. ETC

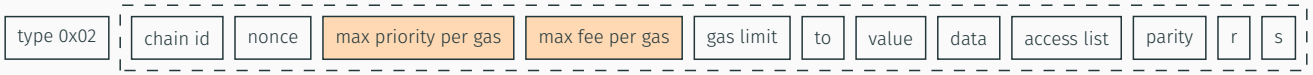
Appendix

EIP-4844 blob transaction

Spurious Dragon hardfork (Nov 2016) – legacy EIP-155 transaction

Berlin hardfork (April 2021) – type 0x01 EIP-2930 transaction

London hardfork (August 2021) – type 0x02 EIP-1559 transaction



Cancun hardfork (March 2024) – type 0x03 EIP-4844 transaction

New block fields

`blob_gas_used` and *`excess_blob_gas`* are new in EL block (execution payload).

`blob_kzg_commitments` is new in CL block (beacon block).

There are no blobs in transactions and blocks. Both contain blob references only. There are blob hashes in a transaction. There are blob commitments in a beacon block.

New precompile and opcodes

New precompile *`point_evaluation_precompile`* takes versioned hash + x + y + KZG commitment + KZG proof, and checks 1) whether a versioned hash corresponds to a committed polynomial and 2) whether a proof proves that $f(x) = y$.

It is useful for ZK-rollups. It can check whether a correct blob was used as a input of ZKP system efficiently. By the way, precompiles for BLS12-381 (EIP-2537) are planed in next hardfork, Pectra.

New opcode *`BLOBHASH`* pops a blob index, and pushes a blob versioned hash on the stack.

The *`BLOBBASEFEE`* opcode of EIP-7516 (in dencun hardfork) allows smart contracts access blob base fee of the current block.

KZG commitment and proof 1/2

A blob consists of 4096 field elements (each 32 bytes). We convert a blob into a polynomial by Lagrange interpolation: $f(0) =$ first element, $f(1) =$ second element, $\ldots$, $f(4095) =$ last element.

Over 141k people participated in KZG ceremony last year, and generated g_1^s, $g_1^{s^2}$, ..., $g_1^{s^{4095}}$, and g_2^s, $g_2^{s^2}$, ..., $g_2^{s^{64}}$. Joonkyo and Hyunggi contributed czg-keremony at that time. We can compute $g_1^{f(s)}$, and it is the KZG commitment of a blob. You can regard KZG commitment as a blob's hash.

If you want to prove $f(x) = y$, calculate a KZG proof π. Anyone can verify that $f(x) = y$ efficiently with KZG commitment and proof without evaluating a polynomial costly. And KZG proofs are aggregatable and ZKP-friendly, too.

x is a random point ideally. In practice, we use the hash of prefix + blob + KZG commitment as x.

KZG commitment and proof 2/2

KZG proof $\pi = g_1^{\frac{f(s)-y}{s-x}}$

verify a ZKG proof: $e(\pi, g_2^s/g_2^x) = e(\text{commitment}/g_1^y, g_2)$

$$
\begin{aligned}
e(\pi, g_2^s/g_2^x) &= e(g_1^{\frac{f(s)-y}{s-x}}, g_2^s/g_2^x) \\
&= e(g_1, g_2^{s-x})^{\frac{f(s)-y}{s-x}} \\
&= e(g_1, g_2)^{\frac{f(s)-y}{s-x}\cdot(s-x)} \\
&= e(g_1, g_2)^{f(s)-y} \\
e(\text{commitment}/g_1^y, g_2) &= e(g_1^{f(s)}/g_1^y, g_2) \\
&= e(g_1^{f(s)-y}, g_2) \\
&= e(g_1, g_2)^{f(s)-y}
\end{aligned}
$$

pairing: $e(a^2, b) = e(a \cdot a, b) = e(a, b) \cdot e(a, b) = e(a, b)^2 = e(a, b^2)$

Blob terms 1/3

blob (128 kb) blob itself

KZG commitment (48 bytes) convert blob to polynomial, and evaluate at trusted setup point
CL's blob reference

KZG proof (48 bytes) convert blob to polynomial, and evaluate at the point that is determined by blob and KZG commitment

versioned hash (EVM-friendly 32 bytes)
[version prefix (0x01) + hash of KZG commitment]
EL's blob reference

Blob terms 2/3

network form [RLP-encoded array of blob transaction + blobs + KZG commitments + KZG proofs]
EL's representation

blob bundle [JSON object of KZG commitment + KZG proof + blob]
from EL to block proposer CL

blob sidecar [SSZ-encoded container of beacon block root + blob index in block + blob + KZG commitment + KZG proof + signed beacon block header + KZG commitment inclusion proof]
CL's representation

Blob terms 3/3

KZG commitment inclusion proof used in the blob sidecar

[Merkle proof of blob KZG commitments field in a beacon block on the top of Merkle proof of this KZG commitment among KZG commitments]

proves that this KZG commitment is contained in this beacon block.

useful for light clients.

this proof lacks left-rightness because blob index tells on which side the node is.

blob KZG commitments field has the SSZ list type, and SSZ Merkleization mix_in_length hashes the list Merkle root with its length (i.e. the number of blobs in a block).

Step-by-step (Thanks, Youngbin :)

user (L2 sequencer) calls *`eth_sendRawTransaction`* JSON-RPC method, uses *network form*

RPC server ⇓

p2p transfer by *`NewPooledTransactionHashes`* messages, additional info (tx type and size) in eth/68 *`NewPooledTransactionHashes`* message, uses *network form*

EL clients ⇓

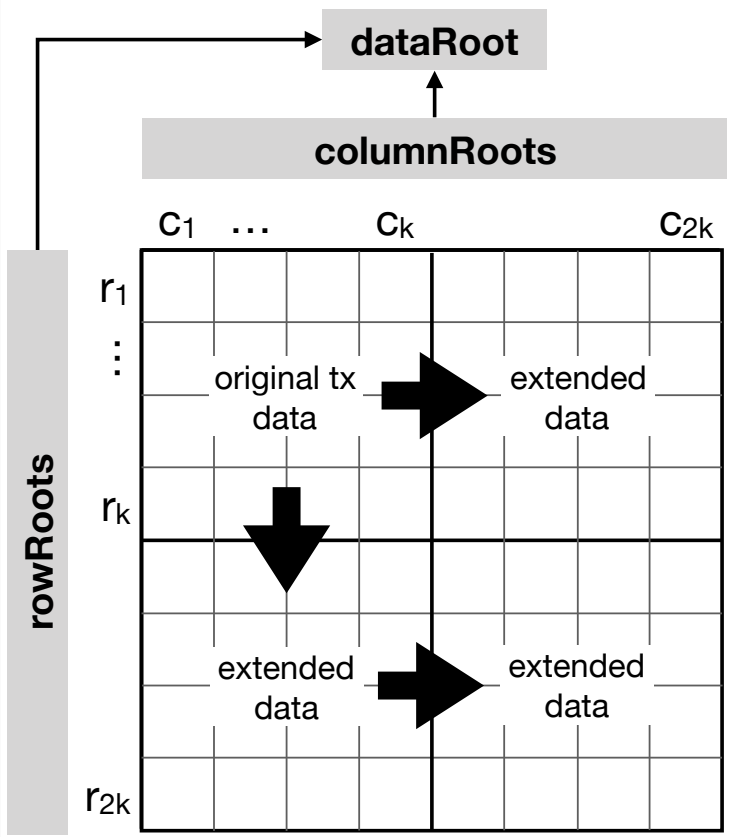
puts *blob bundles* in *`engine_getPayloadV3`* response

proposer CL ⇓

broadcasts *blob sidecars* via gossipsub p2p, or requests blob sidecars explicitly by *`BlobSidecarsByRoot`* and *`BlobSidecarsByRange`*

CL clients check whether blob is available by *`is_data_available(beacon_block_root, blob_kzg_commitments)`*

DAS



2D Reed-Solomon encoding

Reed-Solomon (RS) codes

TCP/IP network corrects data corruption.
Why use it? To prevent tampering

Attackers should withhold more than 1/4 data to make data unavailable. It can be detected by a small number of probes.

see also PeerDAS, SubnetDAS, LossyDAS, IncrementalDAS, DiDAS, FullDAS designs.