This article is part of an ongoing collaboration between Nethermind Research and Flashbots. Thanks to@dmarz and @swp0x0 for detailed discussions on the idea.

Motivation

Rollup experts have expressed clear desires for based rollup designs to allow for some sort of fast confirmation guarantees before L1 blocks are confirmed. Based preconfirmations provide just that. However, the initial based preconfirmation design does not allow the rollup to capture any of the revenue from block-building. This document outlines a protocol which allows for based preconfirmations, while ensuring the based rollup captures much of the value generated from block building.

Based Preconfirmations Primer

Justin Drake recently proposed based preconfirmations as a way for based rollups to allow L1 proposers to provide economic guarantees that an L2 (L2

is used instead of based rollup

where possible for ease of reading) transaction will both be included and ordered so as to be correctly executed. To do this, an L1 proposer must opt in to becoming a preconfer.

Preconfers issue signed preconf promises

to users to include user transactions in their next L1 block. Let's called these user transactions with preconf promises preconfed transactions

. This requires proposers to be able to force include transactions in blocks (requiring something like inclusion lists in PBS). As many L1 blocks can take place between preconfer blocks, preconfed transactions are given priority ordering over all non-preconfed L2 transactions that have been sequenced on the L1 since the last preconfer block.

To ensure some economic guarantees of inclusion from preconfers, preconfers also opt in to two preconf slashing conditions, probably through a restaking collateral deposit in a smart contract or restaking network. Justin

assumes slashing is achieved with EigenLayer-style restaking

These slashing conditions are:

- liveness faults

: A promise liveness fault occurs when the preconfer's slot was missed and the preconfed transaction was not previously included on-chain.

- safety faults

: A promise safety fault occurs when the preconfer's slot was not missed and the promise is inconsistent with preconfed transactions included on-chain.

Summary

Based preconfs allow for inter L2 block-proposal confirmation guarantees (some confirmation time shorter than the confirmation times of the sequencing chain). This is done by aligning L1 proposers with the L2. Unfortunately, this initial sketch seems to leak all of the L2 block-building value to L1 proposers.

Note:

Based preconfirmations are not the same as BFT preconfirmations, as introducedhere. BFT preconfirmations introduce an off-L1 consensus protocol for the L2, with the earliest confirmation guarantee in this new BFT protocol only when transactions are proposed for BFT consensus. As such, based preconfirmations from an upcoming proposer offer reduced latency confirmations over any consensus-based alternative. This is at the expense of a consensus based confirmation. We believe new terminology is required to delineate these two type of confirmations.

Any further mention of preconfirmations/preconfs in this article should be taken to mean based preconfirmations,as described already at the start of this section.

Value-Capturing Based Preconfirmation Protocol

The goal of this proposal is to charge some amount for the right to be a preconfirmer for a based rollup, with this amount representative of the value to be a preconfirmer (~MEV+tips).

Each L1 proposer is automatically a preconfirmer for the based rollup. L1 proposers must state a Harberger value v

,

which is the value they are willing to burn to become a preconfirmer for their upcoming block. The Harberger value defaults to 0, indicating the L1 proposer does not wish to be a preconfirmer. Otherwise, the preconfirmer locks up the specified Harberger value for their block, in addition to the necessary staking requirements to offer preconfirmations, in line with Justin's original proposal:

A proposer must have the ability to opt in to additional slashing conditions. This write-up assumes slashing is achieved with EigenLayer-style restaking.

Given we have access to restaking services, it may be easier to enforce the Harberger value to be locked in the restaking protocol, although locking it on an L1 smart contract is also possible (whatever solution is chosen requires some reasonably strong CR between L1 proposers being revealed, and their block being proposed).

Core Protocol

Let's assume there is an upcoming epoch (some time in advance) with a known list of L1 proposers ~~P_1,…P_n~~

each with Harberger values v_1,…,v_n

. Our protocol works as follows:

- Initially, each proposer P_i

owns preconfirming rights for slot/block i

.

- Any proposer P_i

owning the preconfirming rights for slot i

is able to pay any amount b \geq v_{i-1}

to take the preconfirmation rights from P_{i-1}

.

- If P_i

buys preconfirming rights from P_{i-1}

for b

, then P_{i}

's new Harberger value v'_i

is

v'_i=v_i+b

.

- If P_{i}

owns the preconfirming rights for P_{i-1}, \ P_{i}

can then choose to buy the preconfirming rights from P_{i-2}

, and so on.

- Generally, if P_{i}

with Harberger value v_i

owns preconfirming rights for slots/blocks [i-k,i-(k-1),…,i-1,i]

, and the proposer P_j

owning preconfirmation rights for slot/block [i+1,...,j]

pays any amount greater than v_{i}

, then P_j

gains preconfirmation rights for all blocks [i-k,…,i, i+1,...,j]

.

- When a preconfirmer is confirmed for a block/sequence of blocks, their Harberger value is sent to the L2 vault (the vault represents any destination address which the L2 ecosystem can utilize).

In the top half of the following diagram, each proposer starts off as a preconfirmer, with corresponding Harberger values. In the lower half, P_5

buys preconfirmer rights from P_4

by bidding greater than v_4

for slot 4.

[

|928px;x380px;

1626×665 71.5 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/29b1f0a7feb2f7df9819b00c7b2d63381428618e.png)

An Auction for Exchanging Preconfirmation Rights

Motivation

We define an auction for exchanging preconfirmation rights which settles deterministically when the proposer set is revealed. Deterministic settlement is need as most proposer and preconfirmation slots are revealed too close to the time in which the slot occurs. This does not leave adequate time for any exchange mechanism requiring active participation and CR. In this article, we can see that compute_proposer_index

, the function for calculating proposers in an epoch, depends on effective balance

, which is only updated at the end of an epoch. As such, the first proposer in an epoch is only notified that they are a proposer at the beginning of their slot.

Auction Description

In what follows, we assume L1 proposers have the capabilities to offer L2 preconfs as a service.[1]

Before being elected, each L1 validator i

defines a preconfirmation valuation curve (PVC),

defined by a function PVC_i()

.

PVC_i()

for validator i

defines the amount that validator i

will pay for each slot […, i -1, i]

(remember, proposer i

can only preconfirm during a consecutive sequence of slots which ends in slot i

). As the exact slot i

to which a validator is elected as proposer cannot be known when setting PVC, each validator will need to set PVC_i()

for each i \in [1,...,32]

.

Given the list of proposers and their corresponding PVCs, and a rule for deciding on the set of preconfirmers could be defined as follows:

1. The proposer of the first slot has a single value defined by PVC_1(1)

.

1. If proposer i+n, n\ \geq 1

owns the rights of proposer i

, then:

Proposer i+n

gains the preconfirmation rights of proposer i

if, given proposer i

owns preconfirmation rights for slots

S_{i}=[i-k,…,i], \text{ for some } k \geq 0

, then:

\sum_{j \in S_{i}} \big(PVC_{i+n}(j)-PVC_{i}(j) \big)>0

.

When the proposer set, and valuation curves are known, it will be possible to deterministically calculate the set of preconfirmers for that epoch. Depending on the complexity of this calculation, it may not be performed proactively on the L1, and rather published by the designated preconfirmers through the L2 network. For simplicity, we envision a list of L1 validators who sign-up to offer preconfirmations. When proposing a block, each of those validators are then required to either prove they are not a preconfirmer, or burn the specified Harberger value for the L2.

Discussion

Centralization

It may be desirable to have many unique L1 proposers as preconfirmers. With many independent preconfirmers, there is less risk of censorship, or any other utility-reducing attacks against the L2. Having many rotating preconfirmers also limits the ability of any one preconfirmer to choose malicious ordering rules (it is not currently clear if ordering rules can be enforced upon the preconfirmer, other than socially).

Owning the preconfirmation rights for consecutive blocks is likely more valuable than the sum of values for owning the rights for two separate blocks. If this is the case, there is a clear incentive for proposers to buy as many preconfirmation rights as possible. Given these rights must be bought in advance (ex-ante), this has potentially high capital requirements, with bids based on expected revenue (introducing risk that the realized revenue is lower than expected). This has the potential to centralize the preconfirmer set to specialized actors. To combat this we can introduce several protections:

- A restriction on the number of slots that can be bought by any individual preconfirmer. With such a restriction, large L1 validators may be able to secure multiple consecutive L1 proposer slots (and as such L2 preconfirmation slots), report these slots as having 0 value, yet prevent proceeding proposers from buying those rights due to the slot number restriction.
- Introduce a tax parameter \epsilon_{tax}

such that requirement 3. in our protocol outline for buying preconfirmation rights is now:

”Proposer i gains the preconfirmation rights of proposer i-(l+1)

, l \geq 0,

if, given proposer i-(l+1)

owns preconfirmation rights for slots

S_{i-(l+1)}=[i-(l+1)-k,…,i-(l+1)], \text{ for some } k \geq 0

, then:

\sum_{j \in S_{i-(l+1)}} \big(PVC_{i}(j)-(1+\epsilon_{tax})PVC_{i-(l+1)}(j) \big)>0

.”

With this restriction, preconfirmers must increase the bid of the preceding preconfirmers by at least (1+\epsilon_{tax})

. \epsilon_{tax}

can also be increasing in the difference between the original slot of the buyer and the slot being bought. This would make it more expensive for proposer i+2 than proposer i+1 to buy slot i.

In the top half of the following diagram, we provide one way in which tax could be implemented. In the example, P_2

and P_5

are required to increase the Harberger values for the slots they are purchasing to some exponential of (1+tax) times the previous Harberger values for those slots. In the lower half of the diagram, a restriction is placed on the number of slots that can be bought.

[

|803px;x446px;

1425×791 82.9 KB

](https://collective.flashbots.net/uploads/default/original/2X/e/e3d03af572305d9e8c680e2d7ee6c3063d6ef058.png)

Neither of these solutions for centralization prevent preconfirmers from directly buying preconfirmation rights from proposers through a delegation service/off-chain agreements. This is not unique to our instantiation of preconfirmations (delegatable preconfirmations inherently suffer from this). If delegation of preconfirmations is expected, the delegation service must ensure competition among preconfirmers so the reported Harberger values remain meaningful. Specifically, if preconfirmers collude and monopolize the preconfirmation service, the Harberger values/retained L2 revenue can be kept low. In our auction outline above, the preconfirm relayer could run the delegation auction and enforce/allow for competition among preconfirmers.

If delegation of a particular slot’s confirmation rite is performed through a trusted preconfirm relayer (and the reported value of preconfirming is also trusted), this could be accompanied by a unique tag which prevents the preconfirmation from being resold.

Do we care about value-retention for the based-rollup?

A philosophical dilemma may be playing on your mind: Do we really care about leaking value to L1 proposers/delegated preconfirmers that are providing a preconfirmation service to the based rollup? I believe that we do.

Sequencers/preconfirmers are only one of a minority of actors in a based rollup. Normal users are intended to form a super-majority on any blockchain, including based rollups. Additionally, based ZK-rollups like Taiko must include ZK-provers on this list. If one actor/set of actors is able to extract all of the value generated within the based rollup ecosystem, this same actor/set of actors can centralize and monopolize other roles, while extracting maximal rents from everyone. Worse still, in the case of L1 proposers who inherit the initial right to preconfirm, these actors are chosen by the L1 and are not necessarily aligned with the goals of the based rollup.

Our protocol with based preconfirmations, following on from the original Alternate PBS design, ensures the value generated by the rollup is first sent to the rollup. This value can then be allocated in proportion to the needs of the rollup, and not just into the pockets of L1 proposers and preconfirmation service providers.

[1] In reality, we may see L1 proposers delegating preconfirmation to a professional preconfirmer, in exchange for a fee. This exchange could be policed and enforced by a trusted preconfirm relayer

or using mutual destruction slashing conditions. A trusted relayer is inline with Justin’s description of based preconfirmations

delegated preconf

: If the bandwidth or computational overhead of issuing promises is too high for a L1 proposer (e.g. a home operator), preconf duties can be delegated (fully or partially) to a separate preconfer. The preconfer can trustlessly front collateral for promise safety faults. Liveness faults are the dual responsibility of the L1 proposer and the preconfer, and can be arbitrated by a preconf relay.

This is not an advocation of trusted relayers, but one possible way to implement delegation…