Hybrid PBS in Consensus Layer

Terence Tsao
Core dev @ Offchain Labs
Before we start...

What this talk is:
- **Consensus layer** interface
- Hybrid PBS risks (today)
  - Latency
  - Fault
  - Censorship
- Mitigations

What this talk is **not**:
- **Searcher & builder** interface
- Your typical MEV talks...
Consensus

Execution client

Validator client

Mev-boost

Relayers

Searchers

Builders

Searchers

Builders

Relayers

Mev-boost

Consensus client

Validator client
Background
Normal block processing

Execution client ➔ Consensus client ➔ Validator client

Execution network ➔ Consensus network ➔ Execution network
MEV block processing
## Today's number

- **Network participation (24h)**: 55.31%
  - % of MEV-Boost blocks relayed in last 24h.

- **Flashbots dominance**: 81.36%
  - % of MEV-Boost blocks relayed by Flashbots.

- **Active relays**: 7
  - Relays that relayed at least one block (Flashbots, Bloxroute Max Profit, Blocknative, Bloxroute Ethical, Manifold, Bloxroute Regulated, Eden).

### Top relays
Relays sorted by number of relayed blocks.

<table>
<thead>
<tr>
<th>RELAY</th>
<th># BLOCKS</th>
<th>TOTAL VALUE (ETH)</th>
<th>AVERAGE BLOCK VALUE (ETH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flashbots</td>
<td>64,033</td>
<td>9.418,484</td>
<td>0.147</td>
</tr>
<tr>
<td>Bloxroute Max Profit</td>
<td>6,212</td>
<td>774,929</td>
<td>0.125</td>
</tr>
<tr>
<td>Blocknative</td>
<td>2,264</td>
<td>164,847</td>
<td>0.073</td>
</tr>
<tr>
<td>Bloxroute Ethical</td>
<td>2,101</td>
<td>185,562</td>
<td>0.088</td>
</tr>
<tr>
<td>Manifold</td>
<td>1,477</td>
<td>140,422</td>
<td>0.095</td>
</tr>
<tr>
<td>Bloxroute Regulated</td>
<td>1,369</td>
<td>189,967</td>
<td>0.14</td>
</tr>
<tr>
<td>Eden</td>
<td>1,271</td>
<td>172.03</td>
<td>0.135</td>
</tr>
</tbody>
</table>

[https://www.mevboost.org/](https://www.mevboost.org/)  Oct 13, 2022
Risk: Latency
Normal block processing

Consensus client

Validator client

Execution client

1. prepare_payload
2. get_payload
3. sign_payload
4. submit_payload
5. broadcast

💻

💻

💻
MEV block processing

1. get_header
2. return_header
3. sign_header
4. submit_header
5. submit_signed_header
6. return_payload
7. broadcast

Relay network
Consensus client
Validator client
MEV block processing improved
publish blocks via beacon node #103

Merged

metachris merged 1 commit into main from be-publish-block 18 days ago

Conversation 4 - Commits 1 - Checks 4 - Files changed 9

metachris commented 19 days ago - edited

Summary

Publish the signed beacon block to the network

closes #100

deployed on https://builder-relay-goerli.flashbots.net/

✅ I have run these commands

- make lint
- make test-race
- go mod tidy
- I have seen and agree to CONTRIBUTING.md

https://github.com/flashbots/mev-boost-relay/pull/103
Do the additional round trips + latencies matter?
### Time to propose block differences (without mev-boost)

<table>
<thead>
<tr>
<th>Types</th>
<th>Time to propose (since start of the slot)</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal block</td>
<td>310 ms</td>
<td>20</td>
</tr>
<tr>
<td>MEV block</td>
<td>1098 ms</td>
<td>20</td>
</tr>
</tbody>
</table>

Network: Goerli (1000 validators). Captured on Macbook Pro. 2.6 GHz 6-Core Intel Core i7. 16 GB 2667 MHz DDR4
2022-10-01 - 2022-10-02
https://github.com/prysmaticlabs/prysm/tree/devcon
Normal block proposal timeline

- Run forkchoice to get head
- Prepare block
- Get payload
- Broadcast block
- Submit attestations
- Submit aggregated attestations
- 1/3 * SECONDS_PER_SLOT
- 2/3 * SECONDS_PER_SLOT
MEV block proposal timeline

- Run forkchoice to get head
- Prepare block
- Get header
  - Sign header
  - Submit blind block
- Broadcast block
- Submit attestations
- 1/3 * SECONDS_PER_SLOT
- Submit aggregated attestations
- 2/3 * SECONDS_PER_SLOT
## Block arrival latency differences

<table>
<thead>
<tr>
<th>Types</th>
<th>Time to arrive (since start of the slot)</th>
<th>Samples</th>
<th>Extra Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal block</td>
<td>1158ms</td>
<td>20817</td>
<td></td>
</tr>
<tr>
<td>MEV block</td>
<td>1624ms</td>
<td>16125</td>
<td>&quot;Illuminate Democratize Distribute&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Powered by bloXroute&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;@builder0x69&quot;</td>
</tr>
</tbody>
</table>

Network: Mainnet. Captured on my NUC at home, From Sep,26-Oct,1 300Mb bandwidth. [https://github.com/prysmaticlabs/prysm/tree/devcon](https://github.com/prysmaticlabs/prysm/tree/devcon)
Submit attestation timeline

Waiting to receive block → Validate block → Validate payload → Run forkchoice for head → Submit attestations

Slot S-1

Submit aggregated attestations

1/3 * SECONDS_PER_SLOT

Slot S

2/3 * SECONDS_PER_SLOT

Slot S+1
Submit attestation timeline if block takes longer

1/3 * SECONDS_PER_SLOT

Waiting to receive block

Validate block

Validate payload

Run forkchoice for head

Submit attestations

2/3 * SECONDS_PER_SLOT

Submit aggregated attestations

Slot S-1

Slot S

Slot S+1
Late blocks getting orphaned
## 50% of the orphaned blocks came from relayers (Sep 17 - Sep 27)

<table>
<thead>
<tr>
<th>Orphaned block</th>
<th>Relay</th>
<th>Validator ID</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4788790</td>
<td>No Relay</td>
<td>405190</td>
<td></td>
</tr>
<tr>
<td>4775168</td>
<td>Flashbots</td>
<td>50483</td>
<td>Stakefish</td>
</tr>
<tr>
<td>4773300</td>
<td>Flashbots</td>
<td>333970</td>
<td>Rocketpool</td>
</tr>
<tr>
<td>4750605</td>
<td>Flashbots</td>
<td>293666</td>
<td>Rocketpool</td>
</tr>
<tr>
<td>4745786</td>
<td>BR, max profit</td>
<td>127406</td>
<td></td>
</tr>
<tr>
<td>4734533</td>
<td>No Relay</td>
<td>371425</td>
<td></td>
</tr>
<tr>
<td>4733889</td>
<td>No Relay</td>
<td>323484</td>
<td></td>
</tr>
<tr>
<td>4726812</td>
<td>No Relay</td>
<td>355127</td>
<td></td>
</tr>
<tr>
<td>4721112</td>
<td>No Relay</td>
<td>138183</td>
<td></td>
</tr>
<tr>
<td>4716243</td>
<td>BR, ethical</td>
<td>13278</td>
<td></td>
</tr>
</tbody>
</table>

**slots**: 4715000 - 4790000, **dates**: sep/17/2022, 8:40:23 AM - sep/27/2022, 6:40:23 PM
Consensus client
Validator client
💰 or 😭
Relay network
❓❓
❓
Execution client
Latency == centralization risk?
Latency == centralization risk?
Takeaways

● Latency **matters** for both hybrid and in-protocol PBS
● Latency can lead to **centralization** risks
● **Hard** to optimize network latency
  ○ Beacon client optimization
  ○ Mev boost optimization
  ○ Relay optimization
  ○ Network configs
Risk: Faults
Where relay faults can happen?
Get header (commit) faults

- Malformed header
- Consensus invalid header
- Payment invalid header
- Non conforming header
Malformed header

- Syntactically invalid
- Invalid structure
- Invalid signature

Can consensus layer client detect?
Consensus invalid header

- Invalid with respect to consensus rule
- Invalid block number
- Invalid block hash
- Invalid transaction

Can consensus layer client detect?

```python
class ExecutionPayloadHeader(Container):
    parent_hash: Hash32
    fee_recipient: ExecutionAddress
    state_root: Bytes32 ☠️
    receipts_root: Bytes32
    logs_bloom: ByteVector[BYTES_PER_LOGS_BLOOM]
    prev_randao: Bytes32
    block_number: uint64 ☠️
    gas_limit: uint64
    gas_used: uint64
    timestamp: uint64 ☠️
    extra_data: ByteList[MAX_EXTRA_DATA_BYTES]
    base_fee_per_gas: uint256
    block_hash: Hash32  # Hash of execution block
    transactions_root: Root
```
Payment invalid header

- Payment doesn’t fulfill the value delivered to the proposer
- Require Execution API support

Can consensus layer client detect?
Non conforming header

- Incorrect gas limit
- Incorrect timestamp
- Incorrect parent hash

Can consensus layer client detect?
Get payload (reveal) faults

- Malformed payload
- Consensus invalid payload
- Unavailable payload

(There’s no fall back for these)
Malformed payload
- Execution payload is syntactically invalid
- Full payload does not match committed header

Can consensus client validate? 🟢

*(but it’s too late....)* 💀

```python
class ExecutionPayloadHeader(Container):
    block_number: uint64

class ExecutionPayload(Container):
    block_number: uint64
```
Consensus invalid payload

- Payload contains invalid txs

Can consensus client validate? 📢

*(but it’s too late....)* 💀

```python
class ExecutionPayloadHeader(Container):
    transactions_root: Root

class ExecutionPayload(Container):
    transactions: List[Transaction, MAX_TRANSACTIONS_PER_PAYLOAD]
```
Unavailable payload

- Relay did not make the payload available
- Relay did not fulfill the commitment

Can consensus layer client detect?
Fallback to execution client

- If get header fails*
  - Produce with local execution client
- If get payload fails
  - Can’t produce with local execution client
  - Can’t double sign
  - Ops!! 💀

- return_header can fail **two** ways. Faults or timeout. Faults are better than timeout.
Mainnet incident #1, Sep 16, Flashbots Relay

- Get payload fault
- Malformed payload
- Damage: 3 blocks missed
Mainnet incident #2, Sep 21, BloXroute Relay

- Get payload fault
- Consensus invalid payload
- Damage: 88 blocks missed

Eyal 阿勇 @eyalmarkov

1/ Our MEV services were down for several hours due to a bug that was not identified in testing. We fully take the blame, and I would like personally to apologize to our validators who missed a slot because of it. We will learn from it and improve our services.

8:58 AM · Sep 21, 2022 · Twitter Web App

We would like to provide a short update as well as an apology in regards to the stability of our MEV services. There was a version of our relay released last night that included a bug in how we check blocks provided by our builders. In turn, this resulted in missed bids for select validators.

This version was deployed and tested in our staging environment as well as on chain testnets. The bug was not present in either environment and thus deployed to mainnet. We have since rolled back to a previous version and stability/performance is back to normal. Moving forward we will be changing the method of releasing new versions that will improve our abilities to catch bugs before they reach mainnet.

We are still investigating the root cause of the issue and will post again once this is clear.

Thank you for your ongoing support
Mainnet incident #3, Sep 28, BloXroute Relay

- Get payload fault
- Consensus invalid payload
- Damage: 15 blocks missed

BloXroute Relays Delivering Invalid Blocks

Overview

Over the course of the day on 09/27 the bloXroute ethereal relay was receiving and returning bad blocks to validators from one of our block builders running an experimental build strategy that resulted in missed slots.

Contributing Factors

This builder was improperly committing transactions and caused improperly formatted blocks. Because the relay was not simulating blocks from known builders these blocks were getting sent to validators causing them to miss slots due to proposing blocks with improper receipts or log blooms. A total of 15 slots were missed due to this issue.

Resolution

This was resolved as quickly as we were able to identify where the issue was located.

The experimental builder was taken offline and ensured that it was no longer submitting blocks to any of our relays.

All of the validators who missed slots were paid for the bids they received from the bloXroute relay.

Additionally

On 09/30 all of the relays now simulate all block submissions regardless of source.

Timeline

Slot first seen: 4790116
Slot last seen: 4791561
Mitigation: circuit breaker

- Beacon client to detect "liveness failure"
- Triggered by missing slots consecutively or period of time
  - Ex: missing 3 slots in a row or 8 out of 32 slots
- If triggered, default to local execution client
- Protect against dominant relay/builder malicious or offline
Mitigation: relay monitor

- Score relays based on **behavior** and **performance**
- **Behavior**: relay follows safety and liveness
- **Performance**: relay has good latency
- Relay monitor exports ratings API (i.e. scorecard)
Feature: bid filtering

- Beacon node to filter out bid amount
  - ex: --builder-profit-threshold <wei value>
- If bid is below threshold then default to engine client
- Unfortunately, the local value can't be retrieved at this moment until further engine-API support
Takeaways

- Stil earlym but we need more robust relays
- We need to hold the relays accountable
  - **Idea**: Monitor missing or orphaning slots, poll relay APIs, tweet if relay misses a slot
- We need infra to monitor relays
- `getHeader` faults > `getHeader` timeout > `getPayload` faults
- Relay quality will improve overtime
Risk: Censorship
Some MEV-Boost relays are regulated under OFAC and will censor certain transactions. Use this tool to observe the effect it's having on Ethereum blocks.

49% enforced OFAC compliance
What are the problems?

- Mev-boost (relay proxy) is neutral
- The UX between mev-boost and consensus client is still early
- Hard for consensus client to filter out censored relay
- Harder to determine which relay is censoring at a given time
Potential solutions

- Active inclusion (mev-boost crList)
- Censorship filtering
- Censorship oracle

**Warning**: these are experimental ideas
Active inclusion

- Proposer expresses intent to relays to force txs into payload
- Relayer presents constraint to block builder
- Proposer only accepts payload which txs are included or full block
- Requires multi-proof for included txs from relayer or builder
- Validation can be done on CL client or mev-boost or relay

**Downsides:** timing, latency, complexity, CL client implementation (mem pool, state), additional engine API requirement, trusting relay... etc
Censorship filtering

- Proposer monitors mem pool for top N gas txs at time M
- Relayer replied back header with proofs that top N gas txs are included or all the txs had higher gas
- If top N gas txs are not included, then CL client defaults to local engine

**Downsides:** CL client has to access mempool. How to synchronize M. Loses inclusion control
Censorship oracle

- Introduce a new actor to police censorship
- Ex: relay monitor
- Mev-boost receiving headers but also query oracle to filter out censored headers
- No change on CL client

**Downsides:** Putting trust to a new actor
How much can we constrain builders without bringing back heavy burdens to proposers?

vbuterin

One natural response to the risks of builder centralization (mainly censorship, but also various forms of economic exploitation) is to try to constrain the power that builders have. Instead of builders having full rein to construct the entire block if they win an auction, builders would have a more limited amount of power. This power should still be enough to capture almost all MUV that could be captured, and it should ideally still be enough to capture other benefits of PBS, but it should be weakened to limit opportunities for abuse.

This idea is sometimes called partial block auctions: instead of auctioning off the right to decide everything in a block, auction off the right to decide some things, where those “some things” could be much more nuanced than e.g. “the builder chooses the first half of the block and not the second” - you could give the builder the right to reorder, prepend, append, and you could even constrain the proposer. This post gets into some possible ways of doing this, and some of the tradeoffs that result.

Inclusion lists

In the inclusion list paradigm, a proposer provides an inclusion list, a list of transactions that they demand must be included in the block, unless the builder can fill a block completely with other transactions.
Unbundling PBS: Towards protocol-enforced proposer commitments (PEPC)

Protecting the proposer and ensuring liveness of the chain are a big part of why PBS is considered to be moved into the Ethereum protocol. Ideally, when the proposer utilizes the services of a builder, there is a contract between parties for the delivery of some goods (valuable blockspace), and the contract is honored atomically:

- Either the contract fails to be made and the goods are not delivered/block content is not published, or
- The contract is successfully made and payment always succeeds, no matter what the party committed to supply the goods does.

This stands in contrast to MEV-boost [2], where a proposer could enter into a commitment with a relay, by signing a block header, after which the relay could fail to publish the block in time, and the proposer is not trustlessly compensated while missing the opportunity to make a block.

But with our version of in-protocol PBS (IP-PBS), we bind ourselves to a very specific mechanism for making these contracts, where there is trustless infrastructure for the proposer to sell off entirely their right of making the block. Amendments exist, such as inclusion lists [3], or increasing proposer agency by letting them build part of the block [4]. Still, few results exist showing that a proposer can be fairly unsophisticated and achieve most of the value their position confers upon them.

As an example, what if there is economic value for the proposer in selling the rights to make their block in advance [3], say 10 slots before? Under IP-PBS, a cartel of builders must honor an out-of-protocol market, where the winner of the blockspace future (perhaps auctioned at slot $n-10$) trusts the winner of the slot $n$ IP-PBS auction to let them make the block. Yet the notion of an IP-PBS "winner" is semantically violated, and the value cannot be achieved by an untrusted proposer. Builder relocation with trusted proposers could also increase the delta between what IP-PBS returns to an unsophisticated proposer and what trusted proposers can achieve, beyond simple latency improvements.

https://ethresear.ch/t/unbundling-pbs-towards-protocol-enforced-proposer-commitments-pepc/13879
Takeaways

● **Who** can censor?

● **Who** can filter out censored txs?

● Use the **Builder API** and provide more ways for out-of-protocol markets to organise. Ex: proposer specifies inclusion list, block prefix, etc.

● **Spectrum** of solutions. **Simpler** solutions have **more trusted** assumptions
Final thoughts

- Censorship resistance is the highest priority asides from scaling and withdrawal
- Hybrid-PBS is our best toolboxes, it allows iterations and we need more experiments before in-protocol PBS
- **Shoutout** to all the teams working on hybrid-PBS (mev-boost, relays, builders...etc)