By Alex Watts

Special thanks to @Julian for poking holes in multiple earlier versions,@JustinDrake and @mikeneuder for gaming this out with me in Paris, Imagine/Snoopy and Doug Colkitt for helping me iterate and for the sandwich example, the Flashbots team who pioneered the entire MEV concept and champion its decentralization, the FastLane team who have built and validated many of these theories over the last year on the Polygon PoS blockchain, and the Polygon team for giving us the opportunity . Thanks also to @MaxResnick for his parallel work on ending the proposer monopoly.

Summary:

The FastLane on Polygon (PFL) approach to MEV on the Polygon PoS blockchain has stopped predatory forms of MEV (IE "Sandwich Attacks"), stopped the centralizing effects of private orderflow, and removed the execution advantage of centralized ("Private") relays relative to the decentralized, public mempool. This is not accomplished without cost; unlike PBS, the PFL system is vulnerable to stake centralization from validators who operate their own vertically-integrated "sandwich" bots. This post analyzes the game theory behind how PFL works and explores how those same concepts could be combined with a new style of ePBS to reduce user predation, increase censorship-resistance, and stop the incentivization of centralized p2p. To accomplish this would require a form of ePBS that has multiple proposers and a consensus-layer source of randomness. While those prerequisites may be distant or technologically infeasible, my hope is that this post will arm other researchers with new concepts to use in their arsenal when combatting centralization.

PFL Background:

FastLane Labs is an "On Chain MEV" company focused on using smart contract logic to decentralize the MEV supply chain.

The dominant MEV protocol on Polygon PoS is FastLane on Polygon (PFL). It was deliberately designed without any sort of private relay to Validators. Although there are many benefits to this approach, such as relay decentralization and Validator security, the primary rationale behind the choice was three-fold:

1. We wanted to strongly disincentivize sandwich attacks against all of Polygon's Users, including those who use the public mempool.
2. We wanted Validators to capture all revenue from private orderflow auctions (OFA).
3. We wanted MEV bots to be able to submit bundles via the public mempool. In the future, we hope that this will allow for Polygon validators to capture MEV without relying on any sort of PFL-managed infrastructure such as the FastLane sentries.

To elaborate on point 3, PFL currently maintains a relay between itself and Searchers, but does so for latency reasons. The MEV auctions would still function with this relay turned off, but the bids would arrive slower.

For more on the architechture of the PFL system, please read the whitepaperhere.

PFL's Sandwich Disincentive:

All transactions in all FastLane MEV bundles are broadcast back into the public memory pool. This allows any Searcher to include the transactions of other Searchers in their MEV bundles.

Consider the following structure of a sandwich attack:

Note that the first transaction from the Searcher - the frontrun - is similar to a "loss leader." By itself, it will lose money. The Searcher only realizes profit when their second transaction is executed. The capacity for these transactions to execute in an "all or none" batch is called "bundle atomicity."

PFL intentionally disrupts the atomicity of MEV bundles. All the transactions in an MEV bundle are broadcast to the public mempool, meaning that other Searchers can use them in the construction of their own MEV bundles. Consider a new party, SearcherB, who also wants to make money. What would they do?

SearcherB will combine SearcherA's frontrun transaction and the User's transaction with their own backrun transaction. This leads to three important conclusions:

1. SearcherB's MEV bundle will always be more profitable than SearcherA's, because SearcherA will always have higher costs than SearcherB due to the swap fees and gas fees of the frontrunning transaction.
2. Ergo, SearcherB will always be able to bid higher in auction and is expected to win the auction over SearcherA.
3. Ergo, because SearcherB's MEV bundle includes a cost to SearcherA (the frontrunning transaction), and because SearcherA cannot expect to win the auction without direct and detectable Validator intervention, the rational action for SearcherA is to simply not attempt to sandwich the User in the first place.

This system has been live on Polygon PoS for roughly a year now, and we still have yet to observe a single sandwich attack succeeding via the FastLane relay or smart contract, although we're certain that one will happen eventually. This observed result is particularly noteworthy due to the disproportionately cheaper cost of gas on Polygon; the minimum profit threshold

for a sandwich attack to be actionable is significantly lower than on other chains, even relative to liquidity differences, meaning their frequency should

be higher here.

Ethereum sandwiches on the left, Polygon sandwiches on the Right:

[

1600×383 180 KB

](https://ethresear.ch/uploads/default/original/2X/9/9f6c839bb8b5fc4d195ed3f9815695a06e3ee61e.png)

The data: https://dune.com/hildobby/sandwiches (credit to @hildobby for putting the dashboard together.)

For a more detailed analysis on the math of sandwich attacks and how disrupted bundle atomicity affects a Searcher's PnL, please see this spreadsheet.

This mechanism of "turn everything into a public auction by broadcasting everything to the mempool

" doesn't just work to disincentivize sandwich attacks - it also realigns all "private" orderflow into an auction for the validator's benefit. The validator is therefore able to leverage their monopoly on the blockspace to capture all revenue that otherwise would be going back to users via OFAs, which no longer work without a private path to block inclusion (the exceptions to this are discussed below).

Notably, there are still sandwich attacks occuring on Polygon PoS, but their occurence appears to be limited to two sources:

1. Blocks from the three validators connected to a more extractive type of MEV relay that isn't affiliated with FastLane. These validators make approximately 1-2% of all Polygon PoS blocks.

2. Liquidity pools on SushiSwap. We are still investigating why a disproportionate number of sandwich attacks (relative to liquidity) are occuring through SushiSwap, but the most likely explanation is that these pools on Sushi may be the only source of liquidity for the sandwiched token, meaning that a single attacker can buy the token and induce bundle atomicity without relying on a private MEV relay.

Sandwicher Safety from Induced Bundle Atomicity via Inventory Monopoly:

One of the most common responses to the PFL's approach to MEV is that "sandwich attacks happened before flashbots." While this is true, it's important to examine how

these sandwich attacks happened and why, thanks to token-sniping MEV bots (E.G. JaredFromSubway), this isn't a concern any longer. The irony couldn't be thicker.

Take, for example, the following "frontrun" portion of a sandwich attack:
https://etherscan.io/tx/0x6073062555c134dbc7ad0a88d4c3bb45f8a5fe9b20df9c061f9ff2dd2edd8968

Note that the "frontrun" transaction was through a "PGA" (Priority Gas Auction). It was in the zero index of the block, also referred to as the top of the block ("ToB"). Note also the trade direction - the attacker purchased

the obscure SFI token that the user also

intends to purchase. Finally, note that the user's purchase was not

the transaction following the frontrun; it was significantly further down in the block and at a significantly lower effective GasPrice.

The attacker's backrun was the 14th transaction in the block:

https://etherscan.io/tx/0xc90c3f98b65cb8e1bc2a6c8ec8b3fac76134fc86695ba8919dfd53e08b772ebd

So why did the attacker spend so much money on gas?

In this example - along with most sandwich attacks that occured pre-Flashbots - the attacker was competing for the "Top of Block" tx slot so that they could have a monopoly on token inventory.

The PFL anti-sandwich mechanism and, to a lesser extent, the mempool's native anti-sandwich mechanism, relies on searchers competing with each other to backrun the user. But in order to perform the backrun, a searcher has to sell the token that the user bought.

Competing searchers would therefore need to have access to liquidity for the token being sold. For highly liquid tokens, this can be through flashloans or flashswaps from other pools… but for illiquid tokens found in only one liquidity pool, only the searcher who purchased the token at the top of the block would be guaranteed the inventory needed to perform the backrun

.

These days, integrated token sniping / MEV bots such as JaredFromSubway will carry these illiquid tokens in their inventory. A relevant twitter thread from @bertmiller does a better job of explaining it than I could. The result is that for potential sandwich attackers using the mempool, bundle atomicity through an inventory monopoly is no longer as safe as it was three years ago, largely because their competitors will often hold these tokens in inventory just to gain a gas advantage for backruns.

And as we've spent the last year demonstrating on Polygon PoS, the best way to stop sandwich attacks is to make them too risky for the attacker.

Issues with Implementating PFL on Ethereum:

As a layer 1 from which other layers inherit security, it is critical that Ethereum is not subjected to the same centralization vector to which PFL is vulnerable: a vertically-integrated validator sandwich bot. To block this, we can repurpose some of the work on "inclusion lists" from Mike and Vitalik.

The most basic version of a PFL-like system on Ethereum would target the following objective:

For a block to be valid, each transaction in it must have been observed in the public memory pool by a threshold of attesters prior to the block deadline.

This accomplishes three goals:

1. It adds significant risk to "sandwich attacks," with the intention of stopping them outright.
2. It makes the decentralized mempool the optimal p2p path for users by removing any boost to execution quality (via blocking sandwiching or trade rebates**) that can be provided from centralized relays that have negotiated "off chain" contracts with trusted builders.
3. It removes the value of "private orderflow" for builders, who would still compete to optimize block value but who would now have a level playing field (IE they all have the same transaction set to build with).

(** In the interest of full disclosure, I should point out that trade rebates and other User-aligned execution outcomes could still be handled using account abstraction and bundling User Operations and Searcher Operations together into a single transaction. In fact, FastLane's current project, Atlas, does exactly that by using smart contract logic to create a trustless "smart" environment for executing operations and intents without leaking value to relays, builders, proposers, or other adversarial actors in the MEV supply chain.)

While these three goals are admirable, a problem arises:

If a vertically-integrated proposer/builder/sandwicher knows that it will propose the next block, it can use its latency advantage to release a sandwich attack to the mempool at 'last call,' while simultaneously using this extra value to win the block auction.

Although I question the likelihood of any large accumulator of stake (coinbase, lido, etc) actually attempting this, it's still a valid concern that must be explicitly addressed due to the importance of Ethereum's role as a base layer from which other execution layers inherit security.

Exploring a Potential Implementation on Ethereum:

Four prerequisites should be in place to nullify the stake centralization vector created by the latency advantage of a vertically-integrated Proposer/Builder/Sandwicher:

1. Multiple proposers.
2. A version of ePBS that requires that the proposer propose the most valuable block.
3. MEV Burn.
4. A source of randomness to determine which proposer proposes the canonical block.

Let's start by establishing that the deadline for a transaction to be observed in the mempool to be valid must occur before

the slot's proposer will be aware that it is solely responsible for proposing the block. This is to preserve the risk element mentioned above; once the potential proposer learns that it is the actual proposer, it must be too late for it to place a "valid" transaction in the mempool.

[

image

927×145 10.1 KB

](https://ethresear.ch/uploads/default/original/2X/f/f45f25e2f05c00c309e17d1ba9b69d4acf47ca0d.png)

A specific form of ePBS that requires the proposer select the most valuable block would be required to prevent the proposer from just proposing their own block each time. This is important because, as mentioned earlier, it is always

more profitable to backrun a user + sandwicher than it is to just sandwich the user. Note that a vertically-integrated Proposer/Builder/Sandwicher could program their smart contract to not execute the frontrun if block.coinbase != tx.origin. By requiring that the proposer take the most profitable block, the proposer would no longer be able to assert that the block.coinbase is their intended one.

MEV burn would be required to prevent eigenlayer-enabled proposer/builder/sandwicher collusion that would trustlessly kickback the revenue from the proposer to the builder/sandwicher. (Shoutout to the unaligned anons who accidentally justified MEV burn while trying to poke holes in PBS <3)

One potential implementation:

1. If >X% of proposers have a tx on their list then it must be included. That would be for censorship resistance.
2. If <Y% of proposers don't have the tx on their list then it can't be included. That would be for sandwich protection and nullifying private orderflow.
3. Importantly, X should greatly exceed Y.
4. The randomly-selected proposer would submit an array of blocks, each of which is checked pursuant to steps 1 and 2. This redues the likelihood of no block being approved.
5. The proposer also submits their own inclusion list. The backup block for the next block becomes all the txs on the proposer's list that weren't in the consensus block.
6. Note that inclusion lists are quite complex, and that many of the concerns and solutions identified by@mikeneuder and @vitalik

in their inclusion list design are relevant to any sort of hypothetical implementation discussed here.

- All the data needed to do perform these checks is already stored in the execution client. It would be straightforward to add an interface / API call to the EL client to perform these checks.

But the requirement for multiple proposers and access to a trustless source of randomness (return to PoW?) make this implementation an extremely

heavy lift. Perhaps even impossible. If that is the case, then my hope is that some of the mechanisms and concepts described in this post may inspire other designs that similarly seek to reduce the incentivization of private orderflow, user predation, and centralized relays.