We have seen a spamming dynamic on chains like Base and Solana in which searchers will fire off very many transactions, each with a very high probability of reverting and only capturing every now and then.

The exact conditions that lead to this aren't super well studied but it is likely driven by some combination of

- cheap gas
- high uncertainty. E.g. because the chain is very low-latency or because mempool is private

Depending on your goals, blocks filled with reverting transactions can be problematic for chains. Presumably there is some limit on throughput coming from the need to limit state growth, limit block size for propagation, fit data into available blobspace or to allow community-run fault provers to keep up. Thus TPS is limited and ideally the limited resources are used in an effective way. I will proceed assuming that the sequencer (block builder, leader etc) throughput is not the bottleneck.

I am also assuming that transactions/the block is simulated before being propagated by the sequencer.

Here I am collecting some ideas to deal with this dynamic. These have all been floating around for a while, but I don't know if they've been written down in a public way before.

1. Separating resources

Idea

We can use up more of the sequencer/builder's resources to protect the actual bottleneck. We do this by offering an opt-in RPC endpoint. Transactions sent to this endpoint can never revert (we can create a new transaction type for this if we want to be more secure). If they revert, they must not be included in the block that is propagated to the network. With regard to ordering, the transactions are treated the same as normal transactions (or "better" - detailed in 2).

Without any additional rules, this endpoint would see an even greater transaction load from searchers who have nothing to lose by sending more transactions. In order to avoid overloading the sequencer, we require each of these transactions to come along with a payment in a payment channel to the sequencer. As long as the fee required is lower than the gas fee, it should make sense for searchers to use this endpoint, letting only successful transactions reach the bottleneck.

Further work/critiques

- How do we deal with gas payments for successful non-reverting transactions? Do we require them to pay gas separately from the payment channel or does the get included with the successful tx?
- One may argue that this is shifting the equilibrium strategy that is less decentralised as there is now a greater trusted relationship with the sequencer not to claim payments without simulating transactions. I am not convinced that this is much stronger than what the sequencer can do already. We can also try to ameliorate this with TEEs down the line.
- How to set the fee is a bit of an open question. This depends on sequencer capacity.

2. Provide more information

Searchers are firing off transactions hoping that an opportunity opens up before their transactions land and that they will be the first access this opportunity. If the sequencer knew what they were after we could change that dynamic.

We introduce a new transaction type (longer discussion by @dmarz) in which users declare some conditions under which the transaction must be simulated - e.g. if the ETH-USD Uni pool price moves. Now these transactions will at least execute when there is a higher chance of them succeeding.

However, we now have a new problem: when a user sends a ETH-USD transaction, the sequencer suddenly has to deal with many pent-up transactions. We can deal with this by only ever executing the highest paying k (K=1 being the simplest).

Assuming this works, searchers are much less likely to fire off transactions hoping that they will land in the prime spot because one of these bidding transactions will be prioritised. It makes more sense to directly compete in the auction (ideally).

One alternative design following the same idea is to expose more real-time information to searchers and run the auction in a "just-in-time" manner. One problem with this (see Jito's mempool) is frontrunning concern, which can be mitigated with MEV-Share like privacy. The other is that doing this kind of processing in the "hot path" can incur additional latency for users. There is a balance to be found between optimal information exposure for auction revenue, latency and user protection.

On the opposite end of the spectrum we findprotorev on Osmosis in which there is a fixed backrunning algorithm that is executed after user swaps. This is simpler albeit likely less effective than an auction. Eventually, a similar approach can be used to provide better routes for users instead of collecting arbitrage.

Further work/critiques

- This idea still needs a bunch more work.
- How do we optimally partition opportunities? Is it just by state access? What kind of a combinatorial auction works for this setting? We can definitely get quite complicated on the auction if we feel like its worth it…
- Searchers may still be incentivised to spam given that the auction winner(s) may not succeed in capturing the opportunity.