Background

There are two use-cases for relayer networks that have been getting attention recently:

1. meta-transactions to ease user onboarding (i.e. not require owning Eth)
2. mixers because the withdraw address may not have Eth to pay for gas

Existing relayer network solutions attempt to create relayer networks from scratch (tornado and GSN). The small number of initial relayers in these networks means that they're doomed to have oligopoly pricing until some point in the future where there's a massive number of relayers in each of them.

Fortunately, there's an actor who's already doing relayer-type activity on Ethereum for cheap: smart contract frontrunners. As has been documented elsewhere, these actors scan the mempool for transactions that are profitable to run and 'frontrun' them indiscriminately, frequently with a very low profit margin.

I present here surrogeth, a system for tricking frontrunners into running transactions.

Experimenting with Frontrunners

To convince myself that frontrunners do in fact pick up profitable transactions profitably and to get an idea of what their 'minimum' profit was, I ran a quick experiment.

I loaded a contract with Eth and exposed a single method that would release some of that Eth as a reward to msg.sender

if it was called with a signature of (reward_released, incrementing_nonce)

by a key that I control (Source code). I then attempted to transact (and get frontrun) with this contract. Sure enough, a number of my transactions were frontrun.

Although this is far from statistically significant and all of this frontrunning happened to be done by a single frontrunner, the minimum profit the frontrunner took on my transactions was ~0.00177 Eth, which is an order of magnitude smaller than a representative fee on tornado.cash.

High-level design

The following diagram shows the entire system:

[

vizgraph-20200213-2241-l644xl.png

1487×968 67.1 KB

](https://ethresear.ch/uploads/default/original/2X/7/721bc5b9eb92aa01a8cf697e1d420d6a1edc6376.png)

Numbered interactions:

1. Client checks registry contract for URIs of broadcasters and appropriate fees
2. Client sends signed data to broadcaster's URI
3. Broadcaster broadcasts transaction to Forwarder Contract to the network
4. Frontrunner payload it can profit from in the mempool
5. Frontrunner frontruns transaction to Forwarder Contract
6. Forwarder Contract calls Application Contract. Application Contract sends relayer fee back to Forwarder, which then refunds msg.sender
7. Forwarder Contract logs successful relay + fee in Registry Contract

Mechanism explanation

Broadcasters are necessary to get the client's signed data to the mempool because the application user may not have any Eth to pay for gas. Because broadcasters are effectively offering their txes up to frontrunners, they need to run as efficiently as frontrunners (likely because they are). Otherwise they have no incentive to advertise their URI in the registry contract.

With even a few capable frontrunners advertising their addresses in the registry, signed data will now be broadcast to the entire frontrunner network. Note that this mechanism assumes that frontrunners will want the edge of seeing profitable txes first so much that they're willing to potentially be frontrun. This seems to be the case intuitively, but is worth validating.

One way to think about how this system functions is that we're restructuring apps that need txes relayed to operate on some piece of signed data from the user:

- in mixers, this is the ZKP proving the deposit
- in meta-transactions, this is signed data demonstrating that the user wants to take an action

When these signed pieces of data hit the mempool, they're free money for whoever is willing to bite.

Next steps

I decided to post this before surrogeth is live so I feel a bit more pressure to finish the remaining pieces:

- deploy the contract and publish the address
- documentation, documentation, documentation

If you're interested in using gasless transactions or want to build an alternative, low-fee UI for tornado.cash, let me know! I'll be at EthDenver and hope to kick the tires on this thing.

Frontrunners can permissionlessly list themselves in the registry, but feel free to get in touch if you want to list yourself and need help

Finally, if you have any other ideas for how we can use frontrunners to our advantage, please let me know. I suspect they could fit into the layer 1.5 picture somehow, but haven't quite figured out how yet.

tg discussion link

(thanks to @barryWhiteHat and @weijiekoh for conversations that led to this)