



Recursive产品文档

一、摘要

二、产品分析

1. Bundler是什么？

2. Bundler有哪些功能

3. 为什么选择做bundler

4. 现有的Bundler方案有哪些？

4.1 Nethermind

4.2 Stackup

4.3 Eth-Infinity bundler

4.4 Biconomy代付网络

4.5 Ethereum Protocol Fellowship #3

4.6 Candide-bundler-and-paymaster-rpc

5. Bundler可能出现的问题

6. 以太坊的PBS架构

7. Layer2相关调研

8. 其他调研

三、产品方案

P2P网络

Mempool

Bundler服务

一、摘要

1. 版本迭代记录

版本号	变更日期	描述	
v1.0	2022.12.27	添加摘要，产品分析	
v1.1	2023.1.02	添加bundler架构图	
v1.2	2023.2.14	添加调研文档	
v1.3	2023.03.07	添加产品方案	

2. 名词解释

Uopool: UserOperation Mempool 用户发送UserOp之后，这些交易会首先被放置进缓存中，等待bundler来获取打包的交易

二、产品分析

1. Bundler是什么？

bundler是ERC4337标准中的一个组件，类似于传统EOA交易中的infra。

ERC4337标准是一个账户抽象协议，完全避免了对共识层协议的改变。这个方案没有增加新的协议功能，也没有改变底层的交易类型，而是引入了一个叫做UserOperation（后文中简称为“UserOp”）的伪交易对象。用户会将UserOperation对象发送到一个单

独的mempool（为避免混淆，后文中使用Uopool来代替）中。有一类角色被称为bundler（可能是矿工，也可能是任何访问UserOp Mempool的用户），他们将这些对象打包成一个交易，调用EntryPoint合约的handleOps函数，将该交易发送至链上。

2. Bundler有哪些功能

Bundler作为UserOp上链之前的中继，主要有以下功能

- 监听UserOp Mempool中的交易
- 模拟运行
 - 这里说的模拟运行本质上是计算运行所需要的gas，Bundler需要计算出执行UserOp所需的gas与执行其他合约代码所额外需要的gas，并确保从Paymaster收取的gas大于等于所消耗的gas成本。
- Bundle UserOperation 数组
 - 将UserOp打包成队列
- 执行入口合约
 - 调用EntryPoint中的handleOps函数

3. 为什么选择做bundler

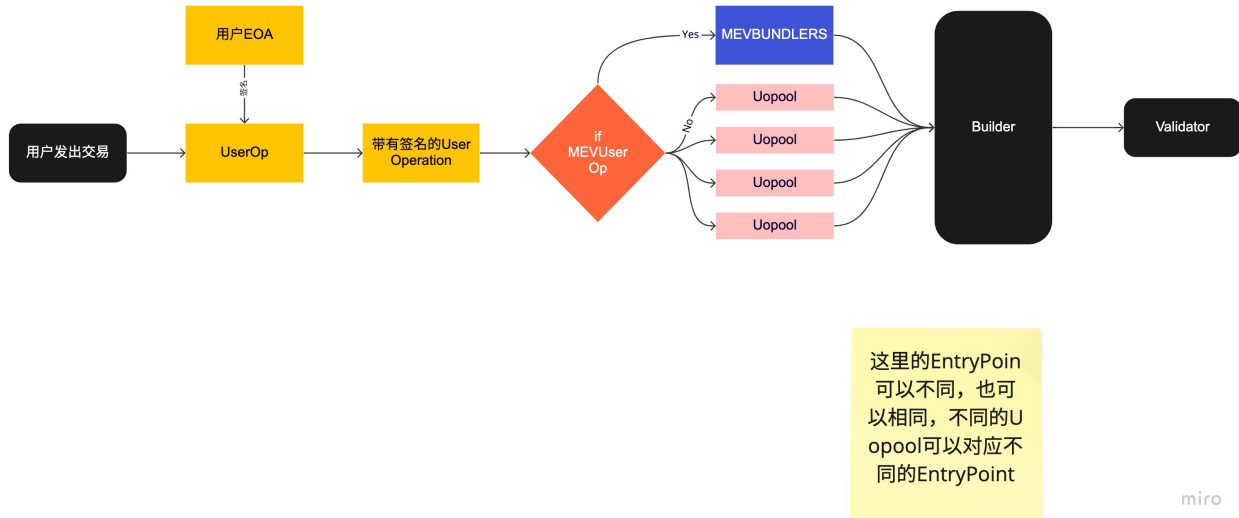
为什么选择做bundler？

4. 现有的Bundler方案有哪些？

4.1 Nethermind

```
https://github.com/NethermindEth/nethermind
```

- Nethermind首先实现了一个非常完善的以太坊客户端，基本可以作为一个工业产品。Bundler实在客户端的基础上加的插件，在配置中可以选择是否开启，所以bundler可以利用客户端的很多已有的特性，比如p2p网络。
- 在客户端实现的好处是，普通的客户端用户也可以接受UserOp并将其转化为Transaction，可以允许某些UserOp只发一笔交易可以通过节点广播到全网，同时，它也支持bundler作为MEVbundler，只接受部分不发送至公共Uopool中的包含MEV的交易
- 支持多个Entrypoint，每个Entrypoint对应不同的Mempool
- 支持paymaster代付功能
- 支持UserOp通过p2p网络的接收跟广播，也支持发现与建立p2p网络的peer连接
- 作为一个客户端，NetherMind拿到UserOp之后，可以广播出去，也可以等到自己出块时，由自己进行打包。



My First Board

https://miro.com/app/board/uXjVP3Y6x_4=/?share_link_id=247759555692

4.2 Stackup

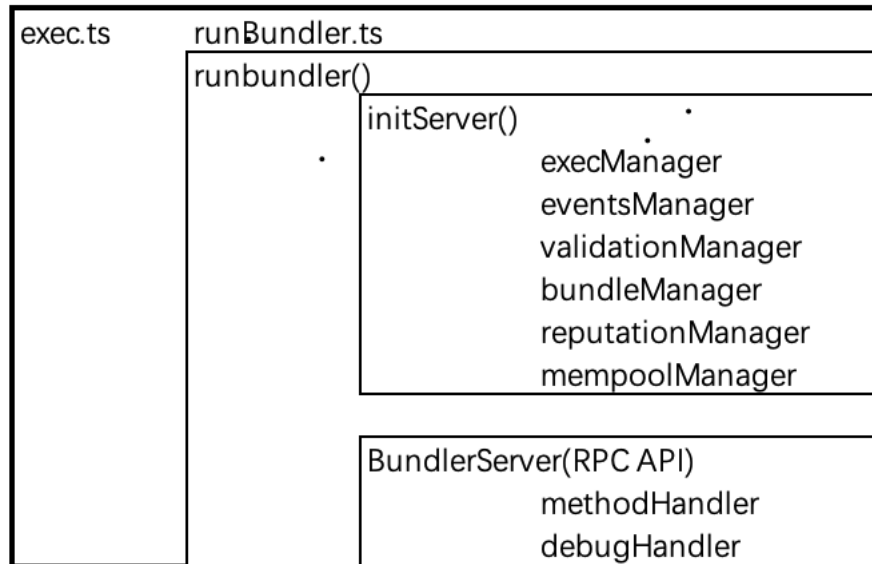
<https://github.com/stackup-wallet/stackup-bundler>

- 这个Stackup实现的bundler是最近看到的最完整，包括基本功能 以及通过debug_traceCall来确定opcode符合EIP
- 虽然他的mempool还是单机本地版本<https://github.com/stackup-wallet/stackup-bundler/tree/main/pkg/mempool> 但是因为对外公开了合适的函数，所以预测它其实是想实现p2p的版本的甚至他的mempool的中继
- <https://github.com/stackup-wallet/stackup-bundler/blob/main/pkg/modules/relay/relayer.go> 在这个部分考虑到了作为bundler可能遇到的恶意攻击
 - 攻击者可以首先发送一个有效的UserOp，然后抢跑这个批次的交易，使得UserOp无效。这种方式会使得Bundler交易失败，并且需要支付交易失败所需的gas，这样会导致Bundler亏损。
 - 也考虑到了作为非出块者可能遇到的恶意攻击，并使用一定的唯一标志（例如IP）来缓解这种情况

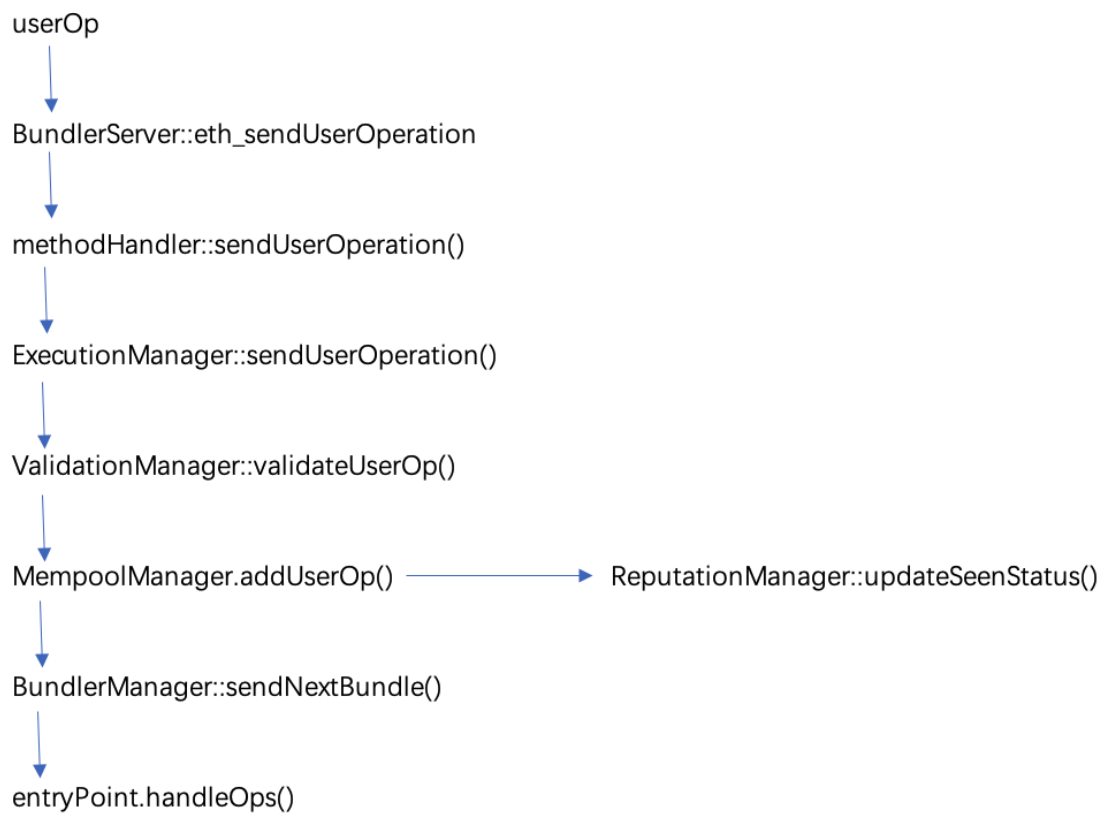
4.3 Eth-Infinity bundler

<https://github.com/eth-infinity/bundler>

启动过程和主要组件：



核心调用逻辑：



主要组件作用：

BundlerServer：

主要功能：

用Express作为httpServer，提供RPC端口调用。

需改进：

1. 用户调用端口跟debug端口在一起，这样就只能开发用，不能用于生产，需分离
2. 没有验证机制、限速机制、黑名单防攻击机制

ExecutionManager：

主要功能：

接收userOp后调用validationManager验证userOp，验证通过后加入memorypool，并根据内部的maxMempoolSize跟autoBundleInterval参数决定是否调用bundleManager让它发送交易。

ValidationManager：

主要功能：

1. 检查userOp的各个参数
2. 检查paymaster、aggregator、deployer是否有充足的stake
3. 检查UserOp, `geth.debug_traceCall (' simulateValidation')`

MempoolManager：

主要功能：

1. 存储userOp的一个数组，用来存储进入的userOp
2. 增加、替换、删除、清空、dump userOp
3. 当增加userOp的时候更新reputationManager的统计数据

需改进：

查找userOp的时候需遍历数组，可考虑改用hash查找；
没有监控新高度然后删除已上链userOp的设计；

ReputationManager：

主要功能：

1. 记录使用aggregator、paymaster、deployer的userOp的出现次数与包含次数
2. 有白名单、黑名单功能
3. 每隔一小时次数减少1/24（间隔在executionManager中设置）

BundleManager：

主要功能：

1. 从mempoolManger获取userOp并取entrypoint执行handleOps
2. 将执行成功的userOp从mempoolManager删除

4.4 Biconomy代付网络

relayer-node/AAConsumer.ts at main · bcnmy/relayer-node
A Multichain Relay Node. Contribute to bcnmy/relayer-node development by creating an account on GitHub.
<https://github.com/bcnmy/relayer-node/blob/main/relayer/src/services/consumer/AAConsumer.ts#L59>

bcnmy/relayer-node

A Multichain Relay Node

3 Contributors

0 Issues

1 Discussion

9 Stars


3 Forks

biconomy原来就有代付网络，他的网络使用的AMQP做的消息队列，他加入4337支持，对于类似bundler模块的支持 仅仅是增加了一个eip4337的消息消费者，这个消费者订阅到消息后 直接使用配置的entrypoint处理，暂时来看也相对比较简单，代码路径：

1. 消息消费者获取4337 OP： <https://github.com/bcnmy/relayer-node/blob/main/relayer/src/services/consumer/AAConsumer.ts#L59>
2. 处理4337交易：<https://github.com/bcnmy/relayer-node/blob/main/relayer/src/services/consumer/AAConsumer.ts#L77>

4.5 Ethereum Protocol Fellowship #3

EIP-4337 Account Abstraction - Bundler implementation in Rust - HackMD
EIP-4337 Account Abstraction - Bundler implementation in Rust The following document provides arc
<https://hackmd.io/@Vid201/aa-bundler-rust>



4.6 Candide-bundler-and-paymaster-rpc

Candide-bundler-and-paymaster-RPC/bundler at main · candidelabs/Candide-bundler-and-paymaster-RPC
You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.
<https://github.com/candidelabs/Candide-bundler-and-paymaster-RPC/tree/main/bundler>

candidelabs/bundler-and-paymaster-RPC

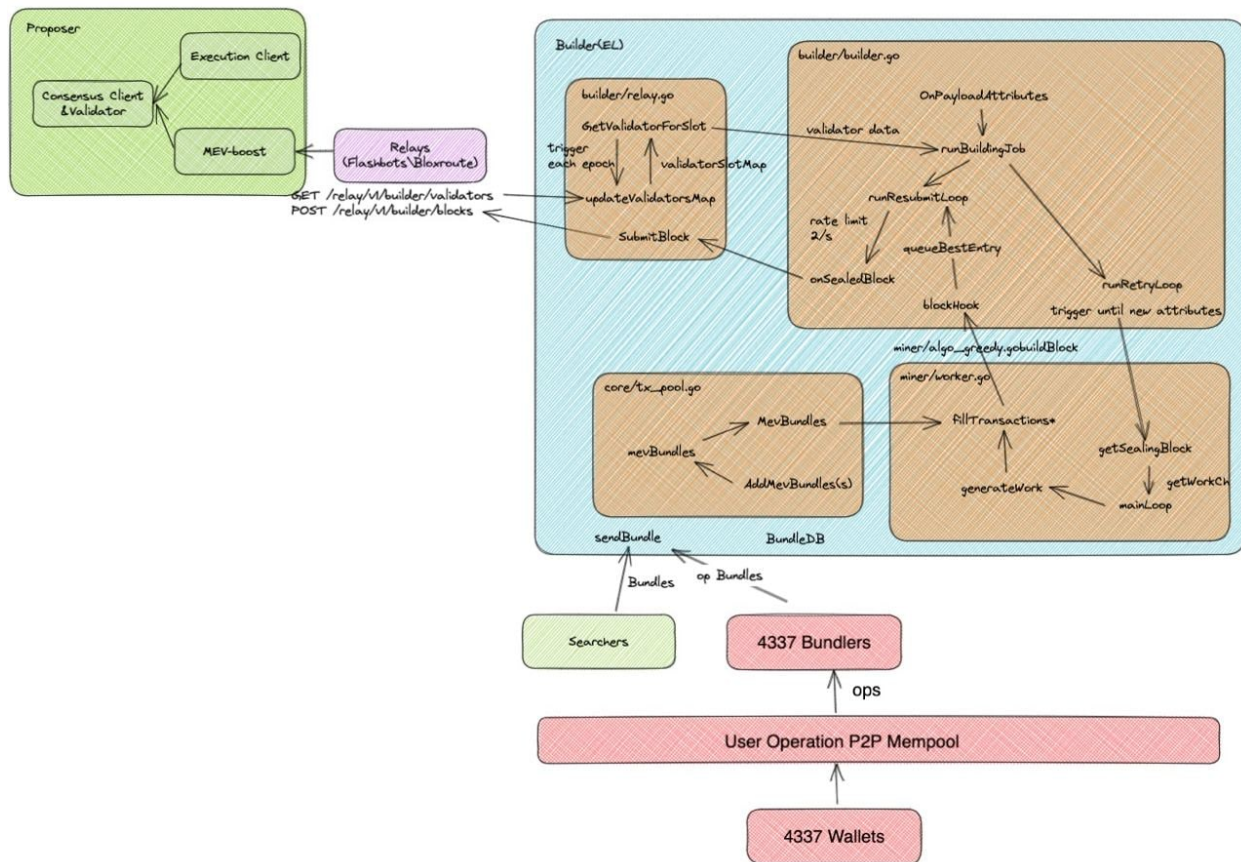
3 Contributors

0 Issues

5. Bundler可能出现的问题

- 抢跑攻击
 - 攻击者可以首先发送一个有效的UserOp，然后抢跑这笔交易，即在执行handlops函数时，付给矿工更高的gas，以此来使得上一个UserOp无效。这种方式会使得被bundler执行的交易无法上链，后续合约Paymaster中补偿bundler的逻辑无法执行，所以Bundler会损失因交易失败而产生的gas
- 修改合约storage以及关联合约的storage导致bundler无法获得足够的补偿
 - Bundler另一个可能的安全隐患是，在验证以及执行上链的过程中，用户可能会修改合约的storage或者是关联合约的storage，导致bundler无法获得足够的补偿。举例来说，如果一个用户发送的UserOp需要消耗1个eth的gas，这时eth的价格为1000u，用户账户内有1000u，Paymaster准备扣除这1000u，以此作为gas支付这笔交易，接下来，bundler在执行上链的过程中代支付了1个eth，然而就在验证交易之后，上链完成之间的这个间隙，eth突然暴涨至2000u，bundler仍然支付了一个eth的gas，但是由于已经验证通过，用户只需要按照之前的逻辑支付1000u即可，bundler净亏损1000u。预言机合约就是上述所说的关联合约，价格变化即为storage修改。

6. 以太坊的PBS架构



7. Layer2相关调研

大部分的Bundler方案都着手于Layer1，研究过程中考虑是否可以搭建一个基于L2的UserOp P2P网络

- <https://notes.ethereum.org/@yoav/unified-erc-4337-mempool>

[Scroll研究](#)

[Optimism研究](#)

[20230203_op_bundler](#)

8. 其他调研

[Flashbots社区关于SUAVE的讨论](#)

[统一的ERC4337mempool](#)

[bundler-spec-tests 分析](#)

https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance 阅读笔记

三、产品方案

产品分为三个部分

1. P2P网络
2. Mempool
3. Bundler服务

P2P网络

- 为什么我们需要公开的mempool和p2p网络？

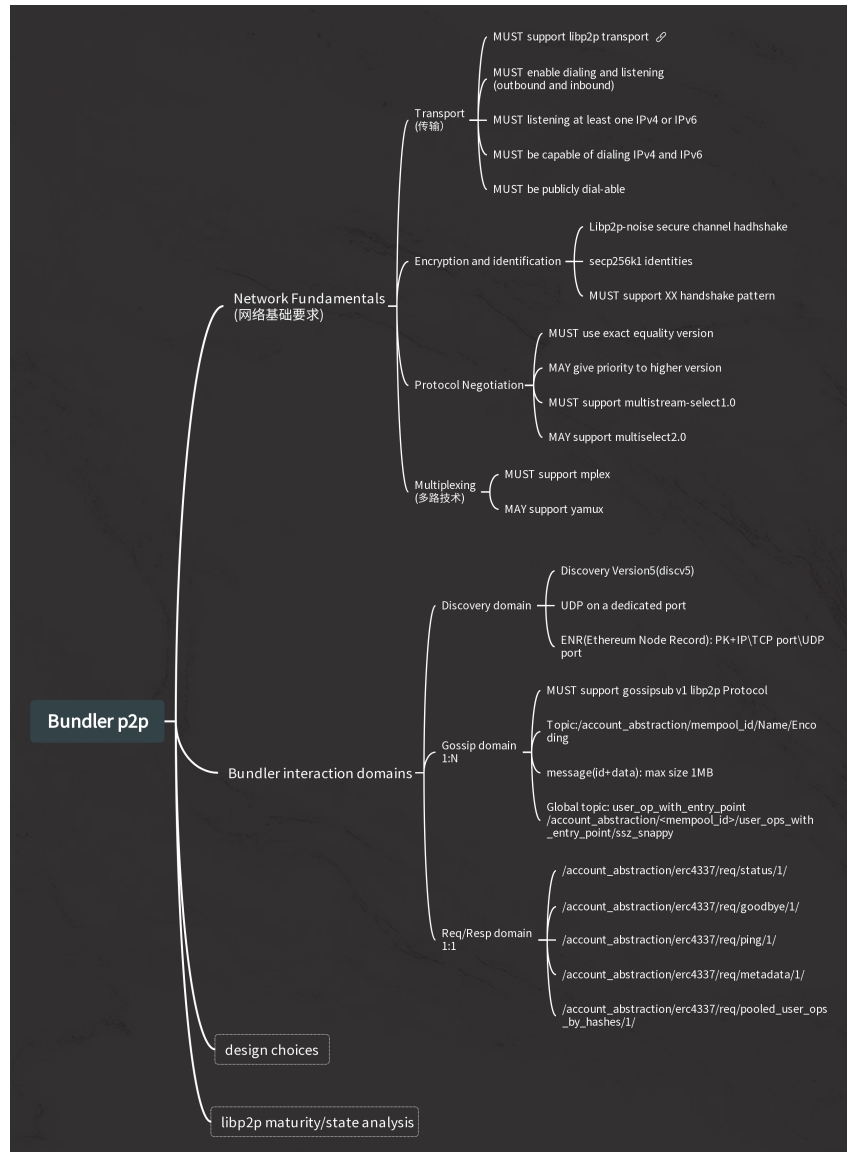
需要public mempool的原因是抗审查。如果一些bundler审查某些特定用户从而不让他们的UserOp上链，那么其他抗审查的bundler会为他们服务，传播并且打包他们的UserOp。公共浏览器（像Etherscan）可以监视public mempool从而确定UserOp被正确的广播了，用户也可以自己运行一个不真正打包的bundler来连接到public mempool并广播自己的UserOp，从而让其他bundler打包自己的UserOp。只要至少有一个抗审查的bundler且用户付了足够高的费用，这个UserOp就会被打包。

Bundler发送Tx的bundle给区块的创建者以让他们的UserOp能被打包上链，但bundle不一定会上链，这取决于Bundler给区块创建者的gas fee，所以对普通的4337钱包用户来说，最好的方案是让他们发送的UserOp能够到达多个相互竞争的bundler而不是只有一个bundler，然后无论哪个bundler获胜，这些UserOp都能被打包。

- p2p网络的标准接口是什么？

<https://github.com/eth-infinitism/bundler-spec/blob/main/p2p-specs/p2p-interface.md>

- p2p网络的标准接口的概要图



• p2p网络的技术实现方案是什么？

按照ERC4337工作组上面定制的p2p interface，整个的标准所使用的技术框架与以太坊共识层节点的基本一致，比如都是基于libp2p库来实现，Network Fundamentals基本一致，Interaction domain也基本一致，主要的区别在于Gossip domain，bundler只需要在节点间广播UserOp信息，所以整个p2p网络都只监听一个Global Topic：user_op_with_entry_point即可。

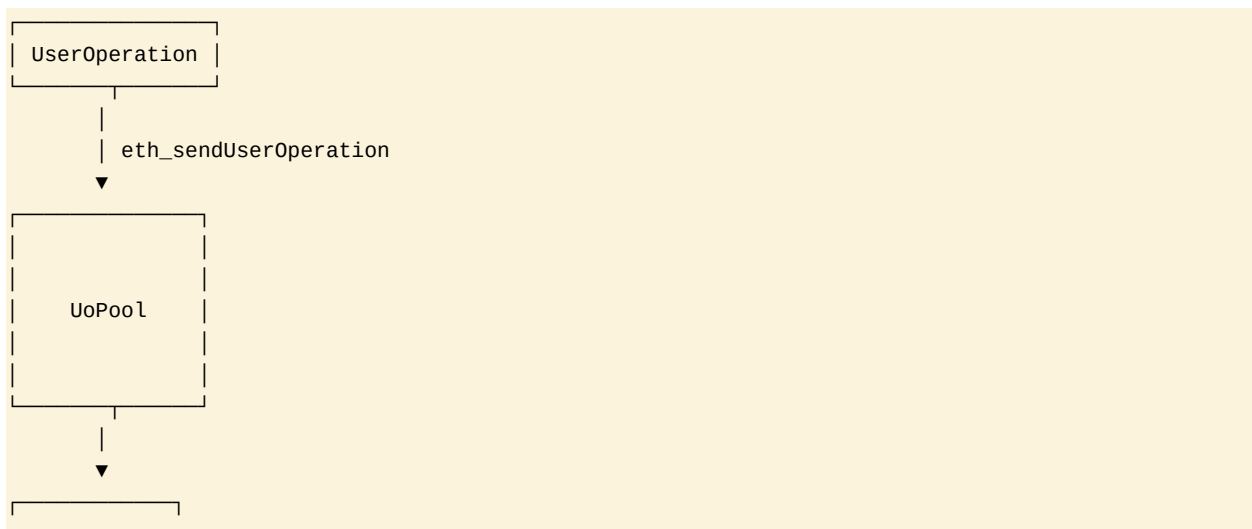
我们可以将已经实现的以太坊共识层的p2p网络部分独立出来，做成一个独立的可执行程序，更改其中需要针对bundler进行优化的部分，与mempool的交互通过内网的RPC端口来实现，这样就可以快速的开发一个独立于bundler程序本身的p2p插件，需要连接公共mempool和p2p网络的bundler可以独立的启动此插件，从而实现读写mempool和与其他遵循同一规则的bundler交互。

可以将PrismLabs的prism client中的p2p部分独立出来，复用大部分的代码，使用golang语言，让它与其他语言（比如java-script）实现的bundler交互。

Mempool

Mempool分为两个部分

- 半公开Mempool，第三方钱包或者应用可以申请加入，需填写基本信息，如产品，官网，联系人等，审核通过后，发送Key。这个功能点
- 对外接口
 - 关键字add，添加UserOp至内存池
 - 关键字get，从内存池中获取UserOp
- UserOp验证功能
 - 单独加入一个验证模块
 - 验证发送交易的账户的费用是否足够，可以支付Bundler执行交易的gas
 - 是否有可能出现其他可能的损失gas的情况（需要讨论）
 - 当UserOp进入内存池后，需要验证UserOp合法性
 - 交易参数是否合法
 - sender
 - nonce
 - initCode
 - callData
 - callGasLimit
 - verificationGasLimit
 - preVerificationGas
 - maxFeePerGas
 - maxPriorityFeePerGas
 - paymasterAndData
 - signature
 - 是否重复
- 除了必要的参数之外，需要第三方提供chainId参数，以便bundler调用对应链的EntryPoint合约
- 监听 EntryPoint events 功能
 - 如果UserOp已被执行，那么从Mempool中删除UserOp



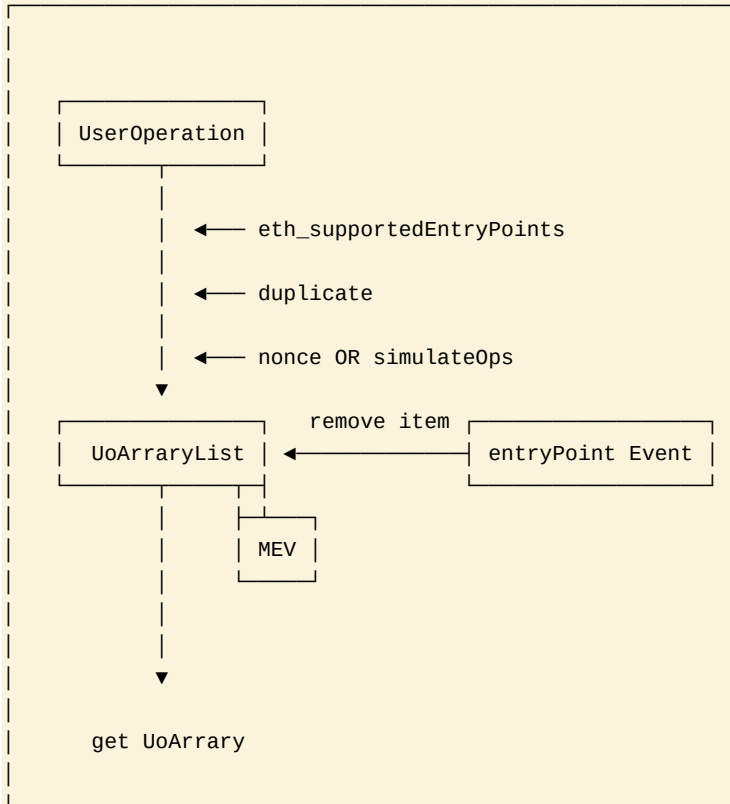
| blockchain |

说明：UoPool为私有内存池（relayer模式只能使用私有池），内存池对外提供一下高级接口：

add：把op添加进入内存池

get：从内存池中获取多个op

UoPool



Bundler服务

即Bundler core与API Endpoint部分所包含的内容

交易发送至Uopool

- 用户通过初始函数（eth_supportedEntryPoints/eth_chainId）确认bundler的状态，API EndPoint是一个HTTP服务，关键是要要做到抗DDOS攻击，可以采用第三方服务
- 接受用户UserOp交易，转发至Bundler Core，这里的bundler core可能会做故障转移恢复，或者是负载均衡。Bundler Core 对UserOp进行验证，确认UserOp合法之后，发送至Uopool，Uopool对交易再做一次验证，同时也需要对外输出一个接口，以至于不同的内存池之间可以互通数据
- Bundler从Uopool中获取UserOp，进行验证，然后执行handleops上链
- 如果监听到UserOp链上状态改变，则在mempool中改变此UserOp状态

