title: Manifold Finance V02 Draft Specification (Infrastructure) author: Manifold finance, Inc. status: public draft version: 2022.05.12

Manifold Finance - SecureRPC/OpenMEV v2.0.0-0 Specification

Context

This serves as a proposed record on the architectural decisions and requirements needed to help facilitate larger network capacity and growth, especially going to multichain capacity. The specification is intended to be a starting point for the development of a v02 of OpenMEV/SeureRPC stack that will be used to connect to the network and provide services in service to staking/vaults.

Consequences

What becomes easier will be being able to attract more developers, also to be able to scale the project to a larger number of users and use cases.

EVM Behavior

We are implementing a new implementation of the EVM for facilitating Constraint-based Speculative Transaction Execution. This is for providing and enriching data, not for actual production of blocks. Additionally, we are also added Instrumentation logic to the EVM this will help for:

- Bug detection
- Implementing a dynamic taint analysis procedure for assessing bugs
- Trace code coverage and data-flow coverage, this is for defensive use.
- Trace comparison operation, for grey-box concolic testing.

Nonstandard and atypical-RPC Methods

Wallet RPC Method supports

eth_newCompletedTransactionFilter

and eth_getFilterChanges

Polling Push notifications require full duplex connections (i.e., websocket or IPC). Instead of subscribing, callers using HTTP send an eth_newCompletedTransactionFilter

request:

{"jsonrpc": "2.0", "id": 1, "method": "eth_newCompletedTransactionFilter", "params": [filter] }

The Ethereum node responds with a filter ID:

{"jsonrpc": "2.0", "id": 1, "result": "0x1"}

When a transaction is submitted, the Ethereum node pushes the transaction notification, including return value, into a queue which is emptied when the caller polls using eth_getFilterChanges

:

{"jsonrpc": "2.0", "id": 2, "method": "eth_getFilterChanges", "params": ["0x1"]}

The node responds with an array of transaction hashes and their corresponding return data, in the order they were computed:

{ "jsonrpc": "2.0", "id": 2, "result": [{ "transactionHash": "0x00000000000000000000000000000000000000000000000000000000deadbeef", "returnData": "0x000000000000000000000000000000000000000000000000000000000000002a" }] }

All transactions that were sealed after the initial eth_newCompletedTransactionFilter

request are included in this array. Again, if the filter param

is a non-empty dictionary (contains either from, to, or hasReturnData

) then only transactions matching the filter criteria generate notifications.

Non-standard RPC Methods that we support or will

This is just a list of the methods we support or will support in the future. If you see a method that has no documentation its because It’s not publicly available, yet.

Flashbotss/MEV RPC Methods

eth_sendBundle

eth_sendMultiBundle

eth_callBundle

eth_callMultiBundle

eth_sendMevBlock

eth_callMevBlock

eth_sendMevTransaction

eth_sendPrivateTransaction

Carrier Transactions

eth_sendCarrierTransaction

eth_callCarrierTransaction

eth_addCarrierGroup

eth_renounceCarrierGroup

eth_addCarrierGroupMember

eth_evictCarrierGroupMember

Relay Switchboard

- eth_setRelayPeer
- eth_getRelayPeer
- eth_getRelayStats

Global Virtual Mempool

- eth_multicall
- manifold_txpool_health

- manifold_txpool_map
- manifold_txpool_search
- manifold_dropTransaction
- manifold_transactionStatus
- manifold_multiCall
- manifold_multiCallGas
- manifold_batchCall
- manifold_batchCall
- manifold_accessList
- manifold_sendTransactionCisBlock
- manifold_sendTransactionTransBlock

Emulation NG-EVM

Logging

EVM Emulator and a data logger for the emulator.

Additionally, we are adding a whole new logging and metrics system to the EVM. This will help for:

- New Trace to log metrics of block listening. (–ratio)
- Add allied nodes to keep connection and broadcast preferentially (–anconfigurl)
- allied nodes provides the following RPC Methods:
- allied nodes provides the following RPC Methods:

Backbone Network will be able to publicly offer its tracing infrastructure to offer rich logging that will help you develop and debug smart contracts and dapps.

This is based off of Hardhat's console.sol

contract. We can inject it into the execution and produce hardhat style logging via RPC, on the fly.

@author Nomic Foundation, Hardhat

eth_sendTransaction Contract deployment: Greeter Contract address: 0x8858eeb3dfffa017d4bce9801d340d36cf895ccf Transaction: 0x7ea2754e53f09508d42bd3074046f90595bedd61fcdf75a4764453454733add0 From: 0xc783df8a850f42e7f7e57013759c285caa701eb6 Value: 0 ETH Gas used: 568851 of 2844255 Block: #2 - Hash: 0x4847b316b12170c576991183da927c2f2056aa7d8f49f6e87430e6654a56dab0

console.log: Deploying a Greeter with greeting: Hello, world!

eth_call Contract call: Greeter#greet From: 0xc783df8a850f42e7f7e57013759c285caa701eb6

Error: VM Exception while processing transaction: revert Not feeling like it at Greeter.greet (contracts/Greeter.sol:14) at process._tickCallback (internal/process/next_tick.js:68:7)

- Emulation Logger

:

There are two kinds of usage scenarios:

1. Logging-In-Time: log the traces of txs and blocks in time to record their real arrival time:
2. `geth ... --emulatorlogger --emulatordir [path of log dir]`
3. Logging the past: log the historical data of blocks and txs. Blocks' arrival time is taken from blocks' timestamp. Txs are taken as arriving between the current block and the parent block randomly. `geth ... --emulatorlogger --emulatordir [path of log dir]`

--generateemulatorlogfrom=[] --generateemulatorlogto=[]

4. Emulator

:

geth ... --emulatordir [path of log dir] --emulatefile [relativePath] --emulatefrom [startNumber of a avaible block]

New config

- –ratio value : External ratio log configuration; 1 for open, 0 for close(default = 1)
- –anconfigurl url: TOML file url from web to config allied nodes

Emulator flags

- –emulatordir=directory: base directory for emulator logger and emulate mode (default value: [datadir]/emulatorLog)
- –emulatorlogger: enable the emulator logger (can also be used in emulate mode)
- –emulatefrom=blockNumber: enable the emulate mode and set chain head in emulation to value
- –emulatefile=relativePath: the log file for emulate mode to read, relative to emulatordir

Emulator log generator flags

- –generateemulatorlogfrom=blockNumber: the start (inclusive) block
- –generateemulatorlogto=blockNumber: the end (exclusive) block

Mempool behavior

When OpenMEV

is disabled

, every sent transaction is added to the virtualized mempool (greypool

), which contains all the transactions that could be mined in the future. By default, Backbone Network’s mempool follows the same rules as Geth. This means, among other things, that:

- Transactions with a higher gas price are included first
- If two transactions can be included and both are offering the miner the same total fees, the one that was received first is included first
- If a transaction is invalid (for example, its nonce is lower than the nonce of the address that sent it), the transaction is dropped.

You can get the list of pending transactions that will be included in the next block by using the “pending” block tag:

const pendingBlock = await network.provider.send("eth_getBlockByNumber", ["pending", false,]);

Transaction ordering

Backbone Network can sort mempool transactions in two different ways. How they are sorted will alter which transactions from the mempool get included in the next block, and in which order.

The first ordering mode, called "priority"

, mimics Geth’s behavior. This means that it prioritizes transactions based on the fees paid to the miner. This is the default.

The second ordering mode, called "fifo"

, keeps the mempool transactions sorted in the order they arrive.

Removing and replacing transactions

See EIP 2831

Transactions in the mempool can be removed using the manifold_dropTransaction method:

```
const txHash = "0xabc..."; await network.provider.send("hardhat_dropTransaction", [txHash]);
```

You can also replace a transaction by sending a new one with the same nonce as the one that it's already in the mempool but with a higher gas price. Keep in mind that, like in Go-Ethereum, for this to work the new gas/fees prices have to be at least 10% higher than the gas price of the current transaction.

Automatic error messages

Acknowledgement to Nomic Foundation and the Hardhat team

Backbone Network should

always know why your transaction or call failed. We aim to provide this information to make debugging your transactions and contracts easier.

When a transaction fails without a reason, Backbone Network should

create a clear error message in the following cases:

- Calling a non-payable function with ETH
- Sending ETH to a contract without a payable fallback or receive function
- Calling a non-existent function when there's no fallback function
- Calling a function with incorrect parameters
- Calling an external function that doesn't return the right amount of data
- Calling an external function on a non-contract account
- Failing to execute an external call because of its parameters (e.g. trying to send too much ETH)
- Calling a library without DELEGATECALL
- Incorrectly calling a precompiled contract
- Trying to deploy a contract that exceeds the bytecode size limit imposed by EIP-170

Supported hardforks

Hardfork

blockNumber

FHomestead

1150000

FTangerineWhistle

2463000

FSpuriousDragon

2675000

FByzantium

4370000

FConstantinople

7280000

FPetersburg

7280000

FIstanbul

9069000

FMuirGlacier

9200000

FBerlin

12244000

FLondon

12965000

FArrowGlacier

13773000

^1

Flashbots/MEV Geth RPC Methods

eth_sendBundle

flashbots standard latest

ethSendBundle

can be used to send your bundles to the relay.

eth_callBundle

eth_callBundle

can be used to simulate a bundle against a specific block number, including simulating a bundle at the top of the next block.

eth_callMultiBundle

eth_callMultiBundle

Multi CallBundle will simulate a bundle of transactions on top of the most recent block. Partially follows flashbots spec v0.5.

// CallBundleArgs represents the arguments for a bundle of calls. type CallBundleArgs struct { Txs []TransactionArgs json:"txs" Coinbase *string json:"coinbase" Timestamp* uint64 json:"timestamp" Timeout *int64 json:"timeout" GasLimit* uint64 json:"gasLimit" Difficulty *big.Int json:"difficulty" BaseFee* big.Int json:"baseFee" }

Manifold Backbone Methods

manifold_dropTransaction

Remove a transaction from the mempool (must be signed by originating account)

manifold_evictTransaction

Remove a transaction from the mempool (admin only)

manifold_setLoggingEnabled

Enable or disable logging in Backbone Network (auth required)

manifold_CreateAccessList

takes eth_callBundle

and adds the arguments:

- SimulationLogs bool
- CreateAccessList bool

eth_multiCall

Implements eth_call

. Executes a new message call immediately without creating a transaction on the block chain.