Ethereum Gas Exactimation

In Ethereum, estimating gas for a given transaction is a tricky problem to solve (especially when attempting to maintainEIP-114 compliance). Most of the well-known Ethereum implementations like Geth[1] and Parity[2] use interval halving (binary search) to estimate gas by running transactions through the EVM until an approximate estimation converges.

AtTruffle , we decided such an approach was an unnecessarily CPU-intensive solution to the problem and set out to find a (theoretically) more performant and perfectly accurate way of estimating gas. The result?Gas exactimation . In this tutorial, we'll go over gas exactimation at a high level in order to demonstrate its precision.

Complete example available atthe following repo .

Prerequisites: You should be familiar with Truffle, Ganache, and Solidity. If you need an introduction please consult the following resources:

- Truffle Quickstart
- Ganache Quickstart
- Solidity Documentation

EIP-114, or the "1/64ths rule"¶

EIP-114 mandates that certain stackdepth-creating opcodeswithhold 1/64th of remaining gas from the stack they create. In practice this means:

1. The gas required for a successful transaction can be greater than the actual gas spent (similar to how gas refunds behave).
2. The extra gas required for a successful transaction varies depending on the transaction's initialgas
3. amount.

A long-standing issue withGanache has been the fact that we haven't returned EIP-114 compliant gas estimations. This has caused our gas estimates to be too low in cases where a transaction executed certain opcodes.Gas exactimation addresses this by considering how the gas withheld at any nested stack depth/frame affects the gas needed outside of its execution context.

Let's see it in action.

Create a New Truffle Project¶

We will usetruffle init to create a new Truffle project and then wire up an exampleSolidity smart contract and test script.

// In a new project directory...

truffle init

Setup an Example Smart Contract¶

With our new project initialized, we will create an example file:ContractFactory.sol .

touch ./contracts/ContractFactory.sol Our file will have two contracts,Contract andContractFactory .ContractFactory will have the methodcreateInstance that we will use to create a new emptyContract .

// ./contracts/ContractFactory.sol pragma solidity ^0.5.0;

contract ContractFactory { function createInstance() public { new Contract(); } }

contract Contract { constructor() public {} } Note thenew keyword being used to create a newContract . A valid statement containing thenew keyword gets compiled to bytecode containing theCREATE opcode which is subject to theEIP-114 1/64th gas withholding.

Write a Test Case¶

Next, we will write aContractFactory test case.

touch ./test/ContractFactory.js This test case will deployContractFactory to a Ganache test network and use a gas estimate provided by Ganache to create a newContract .

// ./test/ContractFactory.js const ContractFactory = artifacts.require("ContractFactory");

```
contract("ContractFactory", () => { it("...should deploy and successfully call createInstance using the method's provided gas estimate", async () => { const contractFactoryInstance = await ContractFactory.new();

const gasEstimate = await contractFactoryInstance.createInstance.estimateGas();

const tx = await contractFactoryInstance.createInstance({
  gas: gasEstimate
});
assert(tx);

}); });
```

A Quick Check¶

Before we run our test, we'll download the most recent version of Truffle that uses Ganachebefore gas exactimation.

// In the project directory...

npm i truffle@5.0.13 And we'll make sure we have the latest version of Truffle installed globally that uses Ganachewith gas exactimation.

npm i -g truffle

Testing Before and After¶

npx truffle test ... ... Error: Returned error: VM Exception while

processing transaction: revert And when we use gas exactimation...

truffle test ... ... Contract: ContractFactory

✓ ...should deploy and successfully call createInstance using the method' s provided gas estimate (130ms) 1

passing (143ms) !!!!

Testing Exactimation¶

But is gas exactimation actuallyexact ?

We'll open our test file and subtract exactly a single unit of gas from thegasEstimate before sending our test transaction.

// ./test/ContractFactory.js const ContractFactory

=

artifacts.require("ContractFactory") ; contract("ContractFactory" , ()

=

{

it("...should deploy and successfully call createInstance using the method's provided gas estimate" , async ()

=

{

const contractFactoryInstance

=

await ContractFactory.new() ; const gasEstimate

=

await contractFactoryInstance.createInstance.estimateGas() ; const tx

=

await contractFactoryInstance.createInstance({

gas: gasEstimate - 1

}) ;

assert(tx) ;

}) ; }) ; Running our test again against Ganache with gas exactimation...

truffle test ... ... Error: Returned error: VM Exception while

processing transaction: revert Exactimation confirmed .

Note: Since the initial release of gas exactimation, an even more performant iteration of the algorithm is currently in review here.