Gas Estimation

The builders developing applications for Titan's ecosystem can use the Titan SDK to estimate the transaction costs in L2.
The cost of an L2 transaction comprises the `L2 execution fee` and the `L1 security fee`. Each fee component is defined as follows. Due to the structure adopted by Titan's Optimistic Rollup, which enhances the security of the L2 network by rolling up L1 transactions to L2, an additional `L1 security fee` exists alongside the `L2 execution fee`.

- `L2 execution fee = L2 gas price * L2 gas used` (L2 gas used must be below L2's Gas Limit)
- `L1 security fee = L1 gas price * L1 gas used * L1 fee scalar`
  - `L1 fee scalar` : The gas cost for the L1 transaction is adjusted using a scaling factor. This allows for dynamic adaptation to current network conditions and optimization of transaction costs, particularly in scenarios where gas prices experience rapid fluctuations.

For more information on L2 transaction fees, please refer to 'L2 fee' section.

Gas estimation using Titan SDK

- In this section, we'll show you how to pre-calculate the gas cost of an L2 transaction with the Titan SDK through an example code. We'll compare the calculated gas cost with the actual gas consumed to see the difference.
- You can find example code for using the Titan SDK directly to calculate L2 transaction gas costs here.
- You can import the Titan SDK and the `ethers` libraries.

```
const ethers = require("ethers")
const titanSDK = require("@tokamak-network/titan-sdk")
```

- Using the `ethers` library, you can create a `provider` and `wallet` instance to generate signatures and send transactions on the L2 network.
  - `l2RpcProvider` : `ethers.providers.JsonRpcProvider` instance for L2
  - `l2Wallet` : Create a wallet for the L2 network using `ethers.Wallet`

```
const l2RpcProvider = titanSDK.asL2Provider(
  new ethers.providers.JsonRpcProvider(endpointUrl)
)
const l2Wallet = new ethers.Wallet(addHexPrefix(privateKey), l2RpcProvider)
```

- We define a function `getEstimates()` to perform gas estimation using the Titan SDK. The `provider` is the Ethereum provider object, and the `tx` is the object of the transaction to be estimated.
  - `estimateTotalGasCost()` : Calculates the total gas cost in Wei by adding the return value of `estimateL1GasCost()` and the return value of `estimateL2GasCost()`.
  - `estimateL1GasCost()` : Calculates the gas cost incurred by L1 and outputs it in Wei units.
  - `estimateL2GasCost()` : Calculates the gas cost in Wei for L2.
  - `estimateL1Gas()` : Calculates the amount of gas used in L1 (L1 gas used)

```
// Get estimates from the SDK
const getEstimates = async (provider, tx) => {
  return {
```

```
    totalCost: await provider.estimateTotalGasCost(tx),
    l1Cost: await provider.estimateL1GasCost(tx),
    l2Cost: await provider.estimateL2GasCost(tx),
    l1Gas: await provider.estimateL1Gas(tx)
  }
}
```

- In this example, we deploy a `Greeter` contract and call a function within the contract to calculate the gas cost incurred on the L2.
- The `Greeter` contract is described below.
  1. `Constructor(string memory _greeting)` : `Greeter` 's constructor. It takes an argument of type string named `_greeting` and stores it in the `greeting` variable. The constructor is executed only once when the contract is deployed.
  2. `function greet() public view returns (string memory)` : A function that returns the value of the `greeting` variable. The `view` keyword indicates that this function does not change the state.
  3. `function setGreeting(string memory _greeting) public` : `setGreeting` function that changes the value of the `greeting` variable. Takes an argument named `_greeting` and updates the `greeting` variable.

```
// Greeter.sol
// SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.0;

contract Greeter {
  string greeting;

  constructor(string memory _greeting) {
//    console.log("Deploying a Greeter with greeting:", _greeting);
    greeting = _greeting;
  }

  function greet() public view returns (string memory) {
    return greeting;
  }

  function setGreeting(string memory _greeting) public {
//    console.log("Changing greeting from '%s' to '%s'", greeting, _greeting);
    greeting = _greeting;
  }
}
```

- You create a factory instance by running the `new ethers.ContractFactory` function with the abi and bytecode of the `Greeter` contract and the L2 signer as parameters. Then deploy the `Greeter` contract. We initialize the instance to be deployed by passing an initial "Hi!" greeting to the constructor.

```
const factoryGreeter = new ethers.ContractFactory(greeterJSON.abi, greeterJSON.bytecode

const greeter = await factoryGreeter.deploy("Hi!")
```

```
await greeter.deployed()
console.log(`Greeter address: ${greeter.address}`)
```

- You can call `greeter`'s `setGreeting` function to change "Hi!" to "Hello!". You call the `getEstimates` function we defined earlier to estimate and calculate the gas costs incurred by L1 and L2. `signer.provider` represents the Ethereum provider, and `fakeTx` is the transaction object to estimate. The result of the estimation is assigned to the `estimated` variable. We can estimate the actual gas consumption by calling the `greeter` contract `setGreeting` function. The estimated gas consumption is assigned to `estimated.l2Gas`.

```
const greeting = "Hello!"

const fakeTxReq = await greeter.populateTransaction.setGreeting(greeting)
const fakeTx = await signer.populateTransaction(fakeTxReq)
console.log("About to get estimates")
let estimated = await getEstimates(signer.provider, fakeTx)
estimated.l2Gas = await greeter.estimateGas.setGreeting(greeting)
```

- You call the `setGreeting` function of the `greeter` instance, passing in the `greeting` to create the actual transaction. Set the gas price for the transaction and execute the `wait()` function. We wait for the transaction to be processed and return the result of the transaction execution when it is complete.

```
console.log("About to create the transaction")
realTx = await greeter.setGreeting(greeting)
realTx.gasPrice = realTx.maxFeePerGas;
console.log("Transaction created and submitted")
realTxResp = await realTx.wait()
```

- For example, the estimated and actual gas costs can be shown below.
  - You can check the total gas cost, L1 gas cost, and L2 gas cost by dividing the estimated gas cost and actual gas cost by the Titan SDK and check the amount of gas consumed by L1 and L2 and the difference between them.