Gas & fees

Ilias

February 10, 2020

Some questions answered in this talk

1. What is **gas**, what are **fees**, what is **burn**? How are they related?
2. How to choose the gas limit?
3. How to choose the fees?

Gas

Block production round, synchronous model

1. injection of operations (txs)
2. propagation on gossip network
3. baker sampling
4. baking
5. block diffusion
   - blocks are re-validated by peers

Block production round, synchronous model

1. injection of operations (txs)
2. propagation on gossip network
3. baker sampling
4. baking $\leftarrow$ **critical path**
5. block diffusion
   - blocks are re-validated by peers

round duration $\geq$ baking time

bounding baking time

baking = applying operations in block on ledger state

Given operations $\{op_i\}_{i=1\ldots k}$, update head's ledger state:

$$state_N \xrightarrow{apply(op_1)} s_1 \xrightarrow{apply(op_2)} s_2 \longrightarrow \cdots \xrightarrow{apply(op_k)} s_k = state_{N+1}$$

Bounding operation application time:

- simple transactions: **constant time, easy**
- contract calls: ???
- contract deployment: ???

Contract calls in Tezos

For **each** contract call:

1. **read** binary repr. of code + contract state from disk
2. **decode** code, storage and input to untyped repr.
3. **typecheck** code, storage and input
4. **run** code
5. **encode** storage to untyped repr.
6. (**write** encoded storage to disk – batched & async)

$\Rightarrow$ bound computation time for 1-6

Looking at existing approaches

- trust the validators: **EOS**
- limited expressivity, bounded size programs: **Bitcoin script**
- proof-carrying code: **Zen**
- gas-instrumented VMs: **Tezos, Ethereum, ...**

1. Manually instrument protocol to consume **gas** s.t.

$$\text{gas} \propto \text{baking time}$$

2. Set per-block & per-op. **gas limit** s.t. e.g. baking time $< 10s$

In practice

At injection time, by default, **tezos-client** automatically picks gas limit

1. sends operation to friendly node
2. node applies operation on ledger state using current protocol
3. returns gas consumption to client
4. client adds 100 "for safety"

$\Rightarrow$ can be overridden with `--gas-limit` client option

if gas limit is too low, **operation will fail** but **fees will be paid**

Takeaway on gas

Gas **protects** against DOS at the **protocol** level

There is a global **per-block** gas limit, as well as **per-op** gas limit

operations **declare** how much gas they want to use:

- decided by simulation
- **or** `--gas-limit`

operation will fail if gas $>$ declared, but **fees will be paid**

Gas cost & limits depends on protocol: **hardcoding limits is bad**

use `--dry-run` to perform simulations

Fees

Fees in a nutshell

1. transactions, calls, originations have a **source** address
2. **source** must attach **fees** ≥ 0 for the baker including those operations
3. baker decides which operation to include in his block

The POV of the baker

Blocks: **limited room** bc of **block** and **per-op** gas limits

$\Rightarrow$ baker has to decide which subset to include in his block

possible strategies:

- altruistic: 0 fees
- greedy: subset that maximizes total fees
- ad-hoc: include only ops from whitelist
- ...

Current default heuristic: approximate greedy strategy

The POV of the user

Need a transaction applied in time interval $[T, T + \delta]$

Pb: how to ensure inclusion when chain is crowded?

Problem of **auction theory**:

minimize fees ensuring transaction inclusion

using partial information:

- past fees
- current fees for pending public transactions

What **tezos-client** does: apply a **hardcoded** conversion rate from gas to **tz**

Takeaway on fees

Fees are paid **to the baker**

There is **no protocol-set relationship between fees and gas**

Fees are set through an off-chain market mechanism

tezos-client will pick fees for you but in a **dumb** way

Proper way to pick fees: estimate fees/gas market price by looking at past blocks and pending txs, then multiply by gas consumption of txs

Use `--fee` client option to override default

Burn

Burn?

Some actions require to **burn tz**:

- originating new contract/sending to empty **tz** addr
- increasing storage usage for a contract

Consumption of **space** on the blockchain

Conversion rate between space and **tz** is **protocol-set**

```
cost_per_byte = 1000 mutez
```

$\Rightarrow$ **do not hardcode**

tezos-client automatically estimates burn by simulation

Wrap-up: gas, fees, burn

Operations consume **gas**, proportional to computational power

Gas is a limited resource per block

Fees are payed by users to bakers to buy slots in blocks, decided by off-chain market mechanism

Users need to **burn** a protocol-set amount of **tz** proportional to space consumption