Peer to Peer Systems and Blockchains

BITCOIN AND THE LIGHTNING NETWORK

Tommaso Colella | Final Term, Part 1 | 05/21/21

Code

The code has been developed and tested using truffle. You can find some more information inside the README.md file, in the code folder.

Gas Cost Estimation

Generic contract functions

These results were obtained on a dummy deployment of the contract, consisting only in a single envelope/vote.

- Gas estimate (**compute_envelope**): 22980 Gas Units
- Gas estimate (**cast_envelope**): 53023 Gas Units
- Gas estimate (**open_envelope**): 139808 Gas Units
- Gas estimate (**mayor_or_sayonara**): 47298 Gas Units

Variations

The following results were obtained by varying the quorum and the nay/yay votes for each contract deployment. In particular, the first result shows a variation with a quorum of 8 voters (all confirming the mayor), the second variation shows a big number of losing voters (due to a very soul-wealthy voter), while the last one shows a balance between the nay and yay voters. The variations reflect on the Gas usage. The comparison with the generic mayor_or_sayonara from above, shows that the smaller the quorum, the cheaper the transaction is. On the other hand, we can also see that having to refund a lot of users results in a very high Gas cost. The balanced votes behavior in terms of Gas usage is expected: only a few of the users must be refunded.

- Gas estimate (**mayor_or_sayonara big quorum**): 73086 Gas Units
- Gas estimate (**mayor_or_sayonara lots of losers**): 153155 Gas Units
- Gas estimate (**mayor_or_sayonara balanced**): 118838 Gas Units

Security considerations on *mayor_or_sayonara*

The biggest security issue with the mayor_or_sayonara function, is the possible reentrancy problem. A proxy contract receiving the refund could be exploited in order to make reentrant calls to the function, possibly withdrawing multiple refunds. Of course, this problem would only occur on an unsafe version of the

function. My implementation tries to avoid this by adding a Check-Effect-Interaction pattern: the *outcome_declared bool* condition is added to voting conditions that are checked by the *canCheckOutcome* modifier, and it is set to true as soon as the function execution begins.

Another possible issue, albeit not strictly related to security, is that the *mayor_or_sayonara* function does not scale very well with the rising quorum, and the gas cost could exceed the block gas limit. The outcome would be quite unpleasant since all the ether of the voters would be stuck on the contract.

Issues with *compute_envelope*

The issue I could find related to the compute_envelope function, is that when it gets called internally from a contract function that uses gas, its input values could easily be seen on the blockchain, since the transaction will be mined in a block, even though the function by itself is pure and does not produce any state-modifying side effect. The only way I see this would be a problem, is the case of a proxy contract that casts the vote on behalf of a user, since it could cost money and potentially it could call compute_envelope from the inside. This is a potential security problem since someone with bad intentions could see the input to the function (sigil, doblon, soul) and anticipate voting intentions. When using web3 to make a call to the function, the problem disappears, since the pure function will only be run on the node to which we are connected, and a transaction will not be generated.