Ethereum Smart Contracts - Lottery Application

1st Aksel Vincent Berg
Barcelona School of Informatics
UPC
Barcelona, Spain
aksel.berg@est.fib.upc.edu

2nd Leonard Halling
Barcelona School of Informatic
UPC
Barcelona, Spain
leonard.halling@est.fib.upc.edu

***Abstract*—This document conveys the methodology used and results obtained as a result of the problem stated as evaluation of a blockchain platform. It will guide the reader through the process of implementing a decentralized lottery application built upon smart contracts and deploying it on a blockchain, in this particular case Ethereum. This process involves obtaining required knowledge of various different technologies, programming languages and frameworks, where the common denominator is the aspect of decentralization. By studying this process, the members of the team obtained several interesting findings regarding evaluating the Ethereum blockchain. In addition, the team also investigated the possibilities of connecting a front-end graphical user interface to the blockchain implementation logics in order to create a fully functional system that end users can interact with. Consequently, both authors experienced a strong learning curve about the aspiring field of blockchain technology, which will be explained further below.**

***Index Terms*—Ethereum, Solidity, smart contracts, Truffle, Ganache**

I. Introduction

Blockchain is a relatively new technology by many featured as an innovation that is going to revolutionize many industries as we know them today. The technology revolves around a distributed database where each node verifies alterations new information that is inserted by other nodes. [wik(2018)] It has participated as an aspiring part of innovation and is connected to an ever increasing number of use cases. Although there are many critical voices in the community denying its applicability and calling it no more than an inflated buzzword, there is no denying that the technology has received a lot of promotion the last couple of years.

Throughout the numerous engaging and informative lectures and discussion session hosted in DS-MIRI during this spring - some of which taking upon blockchain technology - the authors were convinced to investigate the subject further. The members of the team therefore converged to a solution in which concerns utilisation of blockchain technology. This is because it introduces new technology which again creates opportunities and potential innovative solutions for an unknown number of different use cases. These opportunities are regarded by inventive and creative corporations that wants to participate in the development towards new and original business ideas.

There are numerous ways of evaluating a blockchain platform. Firstly, given the aforementioned increasing overall interest for the subject in the community, there are plentiful of different blockchain platforms to evaluate. Further, the actual evaluation can also be performed in numerous ways. Given some thinking and consideration, the choice of the blockchain platform to evaluate in the end fell to the most natural one, namely the Ethereum blockchain app platform. This is the most reasonable choice considering it is one of the most well-known blockchain platforms there is, and according to coinmarketcap as of 29.05.2018 the associated cryptocurrency Ether is the second-most valued, only beaten by Bitcoin. [coi(2018)] In addition, the Ethereum blockchain platform runs smart contracts, namely applications that run upon the blockchain. Ultimately this makes smart contract application on the Ethereum blockchain the most well-documented amongst the different blockchain platforms, and therefore a safe choice for our prefered platform.

Given the choice of the platform of Ethereum, a reasonable evaluating scheme would be to create an application using smart contracts. Therefore, as the scope of the lab description states, the group propose to implement and evaluate a decentralized application (dApp). The dApp that has been made is a lottery system built upon smart contracts, deployed on the Ethereum blockchain platform.

II. Ethereum and Smart Contracts

A. *Ethereum Blockchain*

Ethereum is a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions. The main principles behind Ethereum are: Simplicity - the protocol should be as simple as possible, even if that means a tradeoff with data storage or time inefficiency, Universality - Ethereum does not aim to provide features, instead, the internal Turing-complete scripting language allows the user to construct whatever they like, Modularity - the parts of the Ethereum protocol should be as separable as possible, Agility - details of the Ethereum protocol can be changed if improvements are found, Non discrimination and Non-censorship - the Ethereum protocol attempts to not actively restrict or prevent specific kinds of usage. [Buterin(2013)]

B. Smart Contracts on the Ethereum Blockchain

The best way to describe smart contracts is to compare the technology to a vending machine. Ordinarily, you would go to a lawyer or a notary, pay them, and wait while you get the document. With smart contracts, you simply drop a bitcoin into the vending machine (i.e. ledger), and your document, drivers license, or whatever it may be drops into your account. The process may look something like this: [Buterin(2016)]

1) An option contract between parties is written as code into the blockchain. The individuals involved are anonymous, but the contract is the public ledger.
2) A triggering event like an expiration date and strike price is hit and the contract executes itself, according to the coded terms.
3) Regulators can use the blockchain to understand the activity in the market while maintaining the privacy of individual actors positions.

III. Decentralized Smart Contract Lottery Application

A. Rules

The rules of the decentralized lottery system are the following:

- Lottery tickets cost 0.1 Ether each.
- Each player inputs certain amount of Ether according to how many tickets they wish to buy.
- When the lottery pot cap (5 Ether) has been reached, the lottery will execute.
- Thus, the chances of winning the pot is based on the amount of Ether inserted by each participant.
- When the winner ticket has been elected and the winner has been decided, the pot will be rewarded to the winner, and all the participants will be notified that the game has ended.
- The lottery is then reset.

B. Why lottery as a dApp?

The reasoning for building exactly a lottery using smart contracts are many. From the developers point of view, this makes for an excellent opportunity of creating a useful and interesting application from scratch using smart contracts. From the end users point of view, it is done in order to increase the expected profit as a function of stake, as compared to a centralized lottery.

The hypothesis when implementing such a system is that, given the nature of its decentralization, will provide a greater odds for winning for each player and thus become much more profitable over time, in contrast to a lottery actualized by a centralized authority. In fact, given a perfectly, decentralized world, if a user bets \$1.00, the expected return is the same, namely the break-even point. This corresponds to the following equality:

$$\text{E[income]} = \text{stake}$$

In centralized lottery systems however - such as Powerball - a user can expect getting back only a mere \$0.86 from betting \$1.00, the rest going to commision. Consequently, there should be obvious reasons for gamblers to participate in such a system built with a decentralized paradigm, as compared to the centralized counterpart.

C. Measures for success

The measures for success will for the scope of this project be able to actually implement and deploy such a system, with no major coherent problems. The level of difficulty of the implementation plays an important part when evaluating the platform. The actual profitability for players should be as high as possible, in other words, as close to equal to the stake as possible. This is, obviously, as profitable as a lottery would get, given that no external money other than the players money is inserted. The application should obviously also be able to (pseudo-)randomly decide which player is the winner, with no way to predict the properties of the future winners. All this should be connected to a graphical user interface, giving the end user a great overall experience. This would mean a lottery application maximizing profitability, while at the same time avoiding downtime, censorship, fraud or third-party interference, as provided by the decentrality.

D. Related work

It should be mentioned that the concept of peer-to-peer gambling applications is no new and disruptive way of thinking. In fact, there exists a number of P2P gambling applications from before, whereas the pioneer might be Cyberdice. [Stajano and Clayton(2008)] Furthermore, gambling applications have been deployed on Ethereum blockchain platform before, including the particular case of lottery. However, this project regards going from zero or little knowledge of smart contract implementation to construct and run a decentralized application, investigating the process and the result along the way.

IV. Resources

The resources used in this project include a variety of languages, frameworks and developer tools. The most critical resources are introduced and explained below.

A. Ganache

Ganache is a personal blockchain for Ethereum development. It allows you to run a local Ethereum blockchain and has a Javascript environment which makes it easy to run tests, execute commands, and inspect state. You can for example see the current status (addresses, private keys, transactions and balances) of the accounts, which simplifies debugging and testing. Ganache comes in either a desktop application, or as a command line interface. In this project, were using Ganache-CLI, the command line interface which allows you to run a local blockchain with a single command. [Tru(2018)]

B. Truffle

Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier. Among other things, Truffle provides an easy way to compile the smart contracts into build files, migrating these contracts into the local test blockchain, or even deploy to a live network. [Tru(2018)]

C. Solidity

Solidity is a high-level language for implementing smart contracts. It is a statically-typed language, and compiles to bytecode that then is executable on the Ethereum Virtual Machine. Solidity allows developers to write the smart contracts.

D. File structure

To give the reader an idea of what a Solidity project developed with Truffle looks like, were presenting the file structure of this project below.

1 Lottery
  1a) Build
  1b) Contracts
1b)1. Lottery.sol
1b)2. Migrations.sol
  1c) Migrations
1c)1. Initial_migration.js
  1d) truffle.js

Build - The directory where the compiled JSON files are created. These files contain all the information needed to interact with the smart contract, such as contract address, callable methods, variables, and bytecode.
Contracts - The directory where the contracts are defined in Solidity. This is where the developer writes the logic and functionality of the smart contracts, which are then compiled to the build directory.
Migrations - Contains the Javascript files which specifies which contracts will be migrated to the blockchain.
truffle.js - Truffle config file, specifying the address of the host, port number, network id, etc. While developing, these parameters are set to target the local Ganache blockchain.

E. Initializing a project

The process of setting up a development environment with Truffle is fairly simple. Once we have installed the necessary resources mentioned above, we can just run a few commands in the terminal to get going.

1) Run truffle init. This creates the necessary files and folders needed for development.
2) Run ganache-cli. This will start the local blockchain with the default parameters: localhost:8545.
3) Configure truffle.js and provide the parameters of the running Ganache blockchain.

After these steps, we are ready to define the smart contracts. This particular project only required one smart contract to be defined. Once a project gets more complex, it is common to define multiple separate contracts which can interact amongst each other. When the contracts have been defined, we can simply compile the files by running truffle compile. Then, when we want to migrate the compiled data into the blockchain, we run truffle migrate. After migration, we are provided with the addresses of the contracts. It is these addresses that a front-end application would use in order to be able to interact with the contract.

V. Results and Discussion

A. Unstandardized

While implementing this project, we stumbled across several dilemmas and questions that hadn't occurred to us beforehand. The fact that Ethereum still is a technology at its early stages became really clear while trying to find solutions to fairly trivial problems. There are still few standards for design patterns and architecture, and for now, it is mostly up to the developer to read up on the subject so much that they feel that they are knowledgeable enough to implement a solution of their own that goes along with the main principles in Ethereum.

B. Profitability

By deploying the application on the Ganache testnet and running numerous tests, some interesting statistics were derived. Primarily, the hypothesis regarding increased profitability compared to using a centralized actor turned out to be true. Test cases using three different players yields that 320.000 wei gets lost to gas calculation when playing one round of the lottery. Given that 1e18 wei corresponds to 1.0 ether, the correlation between the stake and the expected income is:

$$\text{E[income]} = \text{stake} \cdot 0.99999999999999989333$$

In other words, the expected income is approximately equal to the stake, as the hypothesis as well as the measures for success states. Hence, there is a significant difference between the implemented decentralized lottery compared to centralized lotteries in terms of profitability. Players will accordingly achieve huge increases in profit going from legacy lottery systems with a central actor to using this system. By using the aforementioned example of Powerball as a standardized example, this increase corresponds to 16%, or 14 cents more for each euro.

Even though this solution is as close as it gets when it comes to 100% return of stake, achieving it completely will be impossible using this particular paradigm. This is because of the gas cost that is related to making operations on the Ethereum blockchain, which will be discussed further below.

C. Random number generation

Using random numbers in smart contracts is very tricky. Especially if you want to eliminate the possibility of miners cheating the system. In classical computing, we can use the entropy on the computer to generate a random number. For example, we can use the time on the computer, the data in the computer, or the mouse movements of the user to generate a random number. The problem with these techniques is that if you want a random number in the Ethereum blockchain, you will end up with a different number locally for every client. There are several solutions to this problem, but each one comes with its own limitations and drawbacks.

1) Blockhash: The principle behind the proof of work in Ethereum is that the miner hashes a block header with a different nonce until they get a certain amount of zeros in the beginning of the generated random hash. Since we get access to these hash values in Solidity, why cant we just use them as the base for our random numbers?

This is one of the simplest ways to generate a random number in Ethereum, and the tradeoff for its simplicity is that it is not very secure. The problem with this technique is that it gives the miners influence over the random number, since the miners can decide whether to broadcast a block or not. In the latter case, the block will instead be broadcasted as an uncle, which is defined as a stale block with parents that are ancestors of the included block. Since miners get a smaller compensation for uncle blocks, the cost of this attack is (BlockReward - UncleReward) + TransactionFees. And the effect of the attack is that it allows the attacker to re-roll the random number.

2) Trusted third party: Another approach is to use a third party to generate a random number for you. Oraclize is a fairly popular Solidity library which allows the user to do just this. The Oraclize library works by providing methods which sends a request from your contract to another contract, this contract will pick up the request and turn it into a http request which queries random.org. When it gets a response, it will return this response to the original contract as a callback function.

The limitations with using third parties is that first of all, you put your trust in them. In this case, you are trusting both random.org as well as Oraclize. Although Oraclize publishes TLS notary proof to show that the data theyre providing really comes from random.org, we still cant be sure that they only queried random.org once. [ora(2018)] In theory, they could keep trying until they got a number that favors them, and we would never know. Moreover, by trusting these services, youre also becoming dependent of them. So if the third party service are experiencing network issues, or if they shut down their service completely, this will in turn affect your program. Additionally, since decentralization often is the primary intention of the blockchain, it can be argued that the third party approach more or less defeats this purpose.

3) Commitment scheme: The commitment scheme is divided into two phases: the commit phase, and the reveal phase. In the commit phase, each party generates their random number locally, pays a deposit to commit to the scheme, and then publishes the hashed random number. In the reveal phase, each party reveals their random number, and thus they get their deposit back. Ultimately, all the random numbers are XOR:ed to get one random number.

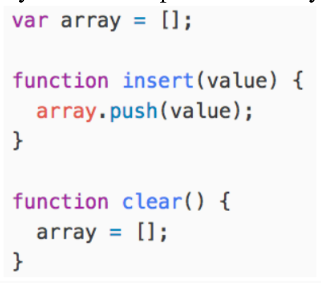
While this approach is arguably better than the previous ones, it also has its drawbacks. First of all, its more expensive to implement in terms of gas. This cost may or may not be significant depending on the amount of participants. A more critical problem is that a party may choose not to reveal their number, this will in turn stop the generation. Furthermore, the last person to reveal can choose whether or not to reveal their number based on the previous reveals, and this will double their chances of winning. In order to discourage this, we need to have a deposit greater than whats at stake in the lottery. This could be very unfeasible for contests with big prizes.

D. Gas cost

Gas is the internal pricing for running a transaction or a contract in Ethereum. Every operation costs gas to run, and the reason for this lies within the fact that in order to run a calculation, this would have to be done in all of the nodes in the entire Ethereum network, which per 29.05.2018 consists of over 16.000 nodes. Likewise, if a contract specifies that a value should be stored in an array, this value must be stored throughout the mentioned computers throughout the blockchain. This makes storing values, in other words changing the state of the entire blockchain - a more costly operation than doing operations or calculations.

As a developer, one would have to consider gas cost when writing smart contracts. Poorly designed smart contracts cost more gas, which in this case would lead to a less profitable lottery. When writing contacts in Solidity, developers can effectively set a gas cap for each method. In this way, it is possible to prevent an application to perform an operation if this operation will cost more gas than one willingly would like to pay. It should also be mentioned that in order to create and deploy a smart contract on the Ethereum blockchain, it is required some onetime amount of gas payment. In the case of the decentralized lottery application, this corresponds to 0.066 Ether. Had the contract been more complex, more gas would have been required.

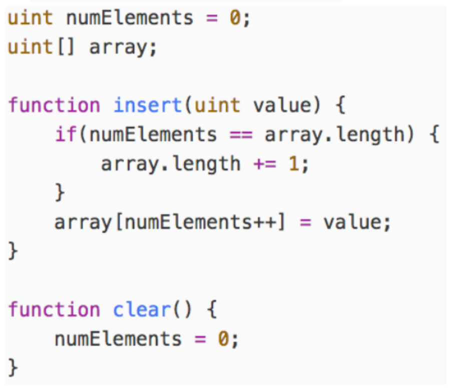
When consider gas cost, developers encounter situations where they have to think in new terms, compared to what they would have done usually. This will often result in code different from usual, but where semantics are staying the same. Below is an example comparing functions operating with an array in JavaScript and Solidity respectively.



```
var array = [];

function insert(value) {
  array.push(value);
}

function clear() {
  array = [];
}
```



```
uint numElements = 0;
uint[] array;

function insert(uint value) {
    if(numElements == array.length) {
        array.length += 1;
    }
    array[numElements++] = value;
}

function clear() {
    numElements = 0;
}
```

Figure 1 shows two functions, one inserting values to an array, and one clearing the same array, as commonly done in JavaScript. Figure 2 shows the exact same semantics, as commonly done in Solidity. As shown, when dealing with an array in Solidity, a good approach will be to manipulate the program into thinking that the array is empty, when it is actually not. In this way, the contracts gas usage is maintained at a lower level compared to simply clearing the array. In fact when looping through lists, if the function returns constant, a very small amount of gas is used. However, if the array is being modified, one should be specifically careful not to run out of gas.

Gas is a severe part of the Ethereum platform and is a necessity to maintain and operate such a ledger. Since it is gas that gives miners the incentive and reward to process a transaction, more gas makes a transaction process faster, while less gas might making the operation end up having to wait a long time before being processed.

Gas will also prevent malicious users from performing attacks such as distributed denial of service (DDoS). Since transactions require gas to be spent for every transaction, gas is spent at every iteration of a loop. This ensures that infinite loop is not possible, since all the gas in an account will end up getting empty after some amount of iteration. When the account is empty, the loop will terminate.

VI. Conclusions

To conclude this project, it is clear that a decentralized lottery application is much more profitable for the player compared to a centralized application. As stated in results, without any commission, the expected income can be approximately equal to the stake.

The implementation of the front-end application in ReactJS was essential for getting an understanding of how a user interface interacts with the blockchain. It was not until we managed to get the website up and running that we fully comprehended how the Ethereum blockchain works.

Considering the scope of the DS course, we were not able to do an actual deploy on the Ethereum blockchain. That would be the next step in this project. Both in order to explore more about gas cost, but also to find out which obstacles that are related with deploying.

Regarding the future, we definitely see a huge potential for smart contracts. What we see as the main objective in Ethereum right now is how we can make it more standardized and more accessible for developers. What we found working on this problem was that the documentation and forums for smart contracts was not nearly as extensive as other, more established technologies.

References

[Tru(2018)] Your Ethereum Swiss Army Knife. 2018. URL http://truffleframework.com/.

[coi(2018)] Top 100 Cryptocurrencies by Market Capitalization, 2018. URL https://coinmarketcap.com/.

[ora(2018)] Oraclize Documentation, 2018. URL https://docs.oraclize.it/.

[wik(2018)] Blockchain, 2018. URL https://en.wikipedia.org/wiki/Blockchain.

[Buterin(2016)] Dmitry Buterin. A Beginner's Guide to Smart Contracts, 2016. URL https://blockgeeks.com/guides/smart-contracts/.

[Buterin(2013)] Vitalik Buterin. Ethereum white paper. *GitHub repository*, 2013.

[Stajano and Clayton(2008)] Frank Stajano and Richard Clayton. Cyberdice: peer-to-peer gambling in the presence of cheaters. In *International Workshop on Security Protocols*, pages 54–70. Springer, 2008.