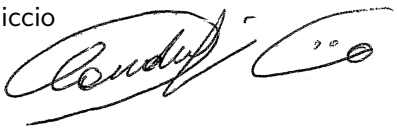
SAPIENZA
UNIVERSITÀ DI ROMA

Estrazione di event log per process mining da ledger Algorand ed Ethereum Classic

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Informatica

Michele Kryston
Matricola 1844733

Relatore
Prof. Claudio Di Ciccio



Anno Accademico 2020/2021

Estrazione di event log per process mining da ledger Algorand ed Ethereum Classic

Tesi di Laurea. Sapienza Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: kryston.1844733@studenti.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Event log extraction for process mining from Algorand and Ethereum Classic ledgers

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Informatica

Michele Kryston
ID number 1844733

Advisor
Prof. Claudio Di Ciccio

Academic Year 2020/2021

Event log extraction for process mining from Algorand and Ethereum Classic ledgers

Bachelor's thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: kryston.1844733@studenti.uniroma1.it

Abstract

Una tecnologia, in ambito informatico, che si sta diffondendo maggiormente negli ultimi anni è quella della blockchain. In parole semplici, la blockchain viene definita come una rete sicura, trasparente e decentralizzata in grado di eseguire transazioni e, nella maggior parte dei casi, anche smart contract.

Un modo utile per comprendere il funzionamento, migliorare ed eventualmente correggere gli errori degli smart contract è l'analisi delle transazioni prodotte. Un possibile metodo di analisi può essere eseguito attraverso il process mining che consiste nel generare modelli di processo a partire da event log. Esistono molteplici applicazioni che si occupano di produrre log a partire da transazioni su blockchain, un esempio di queste è Blockchain Logging Framework (BLF).

Il compito di questo tirocinio è stato quello di aggiungere il supporto di due ulteriori blockchain a BLF, Algorand ed Ethereum Classic (ETC) e verificare la corretta implementazione attraverso lo studio di diversi casi d'uso.

Abstract

One technology that has been gaining in popularity in recent years is the blockchain. In simple terms, the blockchain is defined as a secure, transparent and decentralised network capable of executing transactions and, in most cases, smart contracts.

A useful way to understand, improve and possibly correct errors in smart contracts is through the analysis of the transactions produced. One way of doing so can be through process mining, which consists in generating process models from event logs. There are several applications that deal with producing logs from transactions on blockchain, an example of which is Blockchain Logging Framework (BLF).

The task of this internship has consisted in adding support for two additional blockchains to BLF, Algorand and Ethereum Classic (ETC), and to verify the correct implementation by studying different use cases.

Contents

1 Introduction **1**

2 Background **3**
2.1 Blockchain . . . 3
2.1.1 Security . . . 4
2.1.2 Trilemma . . . 5
2.1.3 Smart contracts . . . 5
2.1.4 Tokens and non-fungible tokens (NFTs) . . . 5
2.2 Algorand . . . 6
2.2.1 Smart contracts . . . 7
2.2.2 Indexer . . . 7
2.3 Ethereum Classic . . . 8
2.4 Blockchain Logging Framework (BLF) . . . 9
2.5 Process mining . . . 10

3 Approach and Implementation **12**
3.1 Approach . . . 12
3.1.1 Algorand . . . 12
3.1.2 Ethereum Classic . . . 13
3.2 Implementation . . . 13
3.2.1 Manifest changes . . . 13
3.2.2 Algorand implementation . . . 15
3.2.3 Ethereum Classic implementation . . . 18

4 Validation **20**
4.1 Simple scenario . . . 21
4.2 Alchemon DApp . . . 24
4.3 Yieldly.finance DApp . . . 27
4.3.1 YLDY asset staking calls . . . 31
4.3.2 Lottery calls . . . 32
4.4 ETCBayc DApp . . . 35

5 Conclusions and Future works **38**
5.1 Conclusions . . . 38
5.2 Future works . . . 38

Bibliography **39**

Chapter 1

Introduction

The blockchain is a public, shared and immutable register in which all transactions of a network are stored. These transactions are grouped into blocks that are linked in chronological order and whose integrity is guaranteed by the use of hash encryption[1]. Transactions are distributed thanks to validator nodes that, in return for a compensation, repeat and validate them. Through the blockchain, in most cases, it is also possible to stipulate contracts between parties in a digital and automated way, so-called smart contracts. What is written in a smart contract is law and, as in standard contracts, must be fulfilled [10].

Transactions executed on the blockchain can be analysed through various methods, one of which is process mining, which simplify the understanding of execution and the discovery of possible issues in the smart contract code [4, 12]. Process mining applications take log files such as XES and CSV as input and generate a model representing the transactions process [17].

One method of extracting data from the blockchain is through the open source Blockchain Logging Framework (BLF) application, which is able to extract transactions from Ethereum [18] and Hyperledger [2]. In order to work, it requires a connection to a node of the blockchain and a manifest file with the information to extract. In output it generates log files, both in XES and CSV format, which can be used for process mining.

My task consisted in implementing two further blockchains on BLF, Algorand and Ethereum Classic (ETC).

Algorand is a blockchain that supports the use of smart contracts and uses TEAL as its language. It was founded in 2017 (2 years after ETC) and uses an innovative consensus method thanks to which it is able to perform more transactions than other blockchains. Extracting transactions from Algorand is facilitated due to the presence of the Indexer which, by taking filters as input, is able to generate the required data information.

Ethereum Classic was one of the first blockchains capable of executing smart contracts and uses Solidity as its language, the same language used by Ethereum since it is a hard fork.

[1]Source: https://tech4future.info/blockchain-cose-esempi-applicazioni/

The remaining part of the paper is subdivided as follows: Chapter 2 discusses the topics, Chapter 3 the applied approach and implementation. Chapter 4 shows case studies, Chapter 5 the conclusion and future developments.

Chapter 2

Background

This chapter explains in more detail the topics covered during the internship which are: blockchain, Algorand, Ethereum Classic, process mining and BLF.

2.1 Blockchain

The blockchain represents a technology that consists of a ledger built as a chain of blocks that, in them, contain transactions. The addition of data in these blocks is chosen through a consensus mechanism, which is governed by all nodes in the network when we talk about *public blockchains* or through nodes authorized to validate the correctness of the data to include in the registry in the case of *private blockchains*.

Its most important features are: register immutability, transaction transparency and traceability and security based on cryptographic techniques. The blockchain is seen as a network and its most prominent property is the management of a database in a distributed manner.

Unlike centralized databases, it offers the ability to update data through the collaboration of network participants and allow the data to be shared, accessible, distributed across all users. So, to summarize the concept, it offers verified and authorized data management without the need for a central authority.

Previously, the blockchain was only identified with *Bitcoin* [13], which is the first blockchain. Later, this identification was also merged with that of Bitcoin *cryptocurrency* which brought confusion regarding blockchain and other areas of development such as *digital currencies*. This led very often to merge blockchain with the concept of alternative or complementary digital currency. In truth, it represents a much broader and articulated phenomenon. Currently, the blockchain is doing through transactions what the Internet did with information and it is doing it thanks to a process that combines distributed systems, advanced cryptography and game theory[1].

[1]Source: https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/

Figure 2.1. Blockchain representation[2]

2.1.1 Security

To secure the network, each block includes a *hash* (a non-reversible algorithmic function that maps a string of arbitrary length to a string of predefined length) that makes each block unique and allows joining with the previous block via identification. To perform a transaction blockchains use *asymmetric encryption* which consists of two keys [16]. The private key used to sign and execute a transaction while the public key to receive it. There are several consensus protocols, the two main ones being *Proof-of-Work* (PoW), an example of which is the Bitcoin blockchain, and *Proof-of-Stake* (PoS) at the core of the Algorand blockchain. PoW protocols perform the mining activity, whereas PoS perform the staking one. Their difference can be summarized as follows:

PoW consists of the computational capacity that users decide to share with the system. The greater the shared computational resource, the more likely it is to make the system stable and to obtain cryptocurrency rewards in exchange for the verification of transactions.

As far as PoS is concerned, the ability to produce rewards is constrained in the validation activity and clients are rewarded in proportion to the cryptocurrency held and placed to guarantee the proper functioning of the system. Therefore, the computational power required for each client to offer security to the chain is stable regardless of the maximum computing power of the client itself[3].

[2]https://www.cbinsights.com/research/what-is-blockchain-technology/
[3]Source: https://www.kriticaeconomica.com/blockchain-algoritmi-di-consenso/

2.1.2 Trilemma

A feature of extreme relevance to blockchains is the *trilemma*. The trilemma is a requirement that addresses the three fundamental principles of the technology i.e. security, scalability and decentralization, which are key features that every blockchain should have. It was coined by Vitalik Buterin (founder of Ethereum) and states that all blockchains can only solve two of the problems just mentioned.

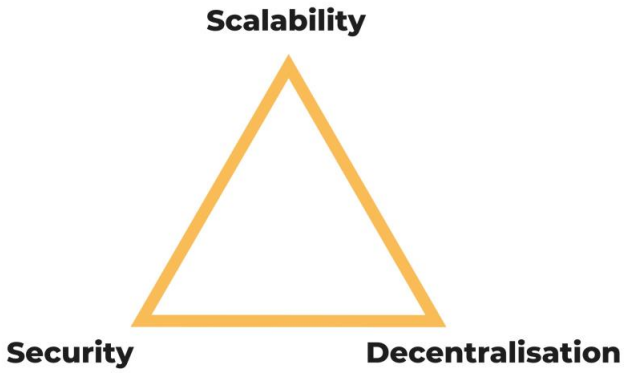


Figure 2.2. Blockchain trilemma[4]

2.1.3 Smart contracts

One of the most important revolutions in blockchain was the introduction of smart contracts. They are defined as "computerized transaction protocols that execute the terms of a contract," since, as Nick Szabo (the one who is credited with the creation of smart contracts) says, "by translating contract terms into codes and embedding them in hardware or software capable of self-enforcing them, it is possible to minimize the need for intermediaries between parties and harmful or accidental exceptions" [15].

This revolution has led to the emergence of numerous decentralized applications called *DApps* that have innovated the blockchain with the deployment of tools such as *DeFi* (Decentralized Finance) and *NFTs* (non-fungible tokens). One of the first blockchains to introduce smart contracts was Ethereum in 2015, this innovation has currently made it the second most capitalized blockchain after Bitcoin. Several other blockchains capable of executing smart contracts have been developed since Ethereum, two examples are Algorand and Ethereum Classic.

2.1.4 Tokens and non-fungible tokens (NFTs)

In most blockchains capable of running smart contracts, it is possible to create and distribute additional "coins" called *tokens*. Tokens are recorded digital information

[4]https://investire.biz/articoli/analisi-previsioni-ricerche/bitcoin-e-criptovalute/criptovalute-bitcoin-trilemma-blockchain-cosa-funzionamento-soluzione-lightning-network

representing some form of entitlement like ownership of an asset, access to a service, receipt of a payment and are generated via smart contracts [5].

Unlike, when we talk about *NFT* (non-fungible token) we refer to unique, non-modifiable digital works that are registered on the blockchain and therefore have a unique and certified ownership. Compared to replaceable tokens, NFTs are in no way interchangeable with each other. Examples of NFTs are drawings, gifs, videos and any element as long as it is digital[6]. They are also generated (*minted*) by smart contracts.

2.2 Algorand

Algorand (ALGO) is a blockchain platform founded in 2017 by cryptographer Silvio Micali, focused on high-speed transaction processing and smart contracts for decentralized finance (DeFi). Algorand's goal is to be a network designed and built to be scalable, cost-effective, extensible, and highly programmable [3]. The way it works is through a consensus protocol similar to Proof of Stake called *Pure Proof of Stake* (PPoS). The consensus offers full participation to those in the network, ensuring protection and speed. The use of PPoS is framed in the achievement of the block production speed measured in a few seconds (about 4.5), which is able to offer a high scalability of the network, that allows the manage of thousands of users at the same time, with very short waiting times and with low commission costs.

Furthermore, the team has devised a new consensus algorithms, fault-tolerant Byzantine systems, and a completely new hashing and digital signature system to enable the highest possible scalability. Thanks to these technologies it is able to handle in the current state of the network about 2 thousand transactions per second without any problems[7].

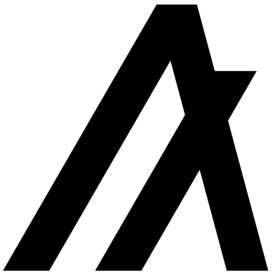


Figure 2.3. Algorand Logo[8]

[5]Source: https://blog.osservatori.net/it_it/token-blockchain-come-funzionano
[6]Source: https://www.ilgiorno.it/economia/nft-cosa-sono-rischi-guadagni-1.7283597
[7]Source: https://academy.bit2me.com/it/cos%27%C3%A8-algorand-qualcosa/

According to Professor Silvio Micali, it is also the only or one of the few blockchains that has managed to solve the trilemma problem.

2.2.1 Smart contracts

Algorand is a blockchain created to take smart contracts to its fullest expression. For this reason, it has established the *Algorand Smart Contracts 1* (ASC1). These smart contracts are designed to leverage the full capabilities of Algorand as a network thanks to the use of *TEAL* or *Transaction Execution Approval Language*.

TEAL is a stack-based language that results in bytecode which is injected into Algorand transactions. Programs can only access in read mode to the transactions they are attached to, to transactions in their atomic transaction group, and to some global values. They also cannot modify or create transactions, only reject or approve them. On average, a smart contract written in TEAL takes about 5 seconds to execute. Another point in favor of TEAL and the way it works is that it is very cheap to run, a smart contract costs about 0.001 ALGO to execute (currently less than 1/10th of a cent)[9].

The blockchain supports also the creation of NFTs and tokens called *Algorand Standard Assets* (ASAs) that are able to maintain the same speed and ease of use as the native ALGO [14].

2.2.2 Indexer

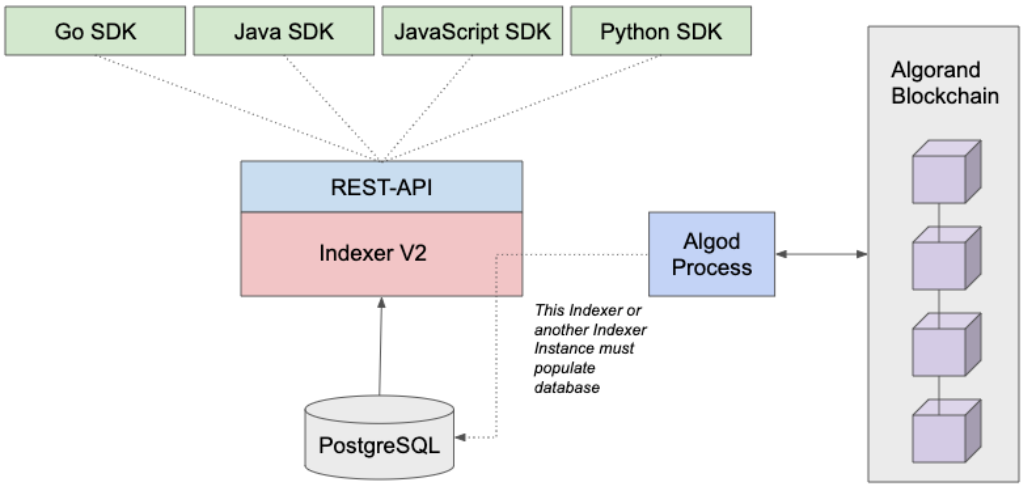


Figure 2.4. Indexer operation diagram[10]

Indexer's main purpose is to deliver a REST API call interface to help with searching on the Algorand blockchain. It fetches blockchain data from a PostgreSQL-compatible database that needs to be filled. This database is populated using either the same

[8] https://cryptologos.cc/algorand
[9] Source: https://academy.bit2me.com/it/cos%27%C3%A8-algorand-qualcosa/
[10] https://developer.algorand.org/docs/get-details/indexer/

instance of the Indexer or a separated one that must connect to the algod process of a currently running Algorand node to retrieve the blockchain information. This node also needs to be an Archival node to enable searching across the whole blockchain. The Indexer delivers also a collection of REST API calls for querying Transactions, Accounts, Assets and Blocks on the blockchain. These calls provide different filter parameters to assist in refining queries [6].

2.3 Ethereum Classic

Ethereum Classic (ETC) was created as a result of a hard fork that split the Ethereum blockchain due to a hacker attack during the year 2016 that managed to steal a total of 3.6 million ETH. It was the launch of *The DAO* (Decentralized Autonomous Organization) project that initiated the disastrous event, that was implemented in the form of a smart contract within the Ethereum blockchain. At the beginning, the project was a great success (in fact, in a very short time it raised about 150 million dollars, equal to 11.5 million ETH), but soon were discovered in its code errors, which made the smart contract vulnerable to theft.

After the theft, which caused a blockade of the entire cryptocurrency community, Ethereum founder Vitalik Buterin and co-founder Gavin Wood decided to make an internal split. The original blockchain was named Ethereum Classic, while the new blockchain kept the original name Ethereum.

Also Ethereum Classic supports smart contracts which allow developers to create and use DApps and issue tokens and NFTs. The ETC blockchain network uses Proof-of-Work as its consensus method, operates via *SputnikVM*, an optimized virtual machine made accessible to low-power devices, includes a node called *Mantis* and the *Emerald* toolkit. The programming language of the ecosystem is *Solidity*, the same one used by Ethereum. Since it is one of the first blockchains to support smart contracts its execution speed is limited to 19 transactions per second, while blocks are generated every about 13/14 seconds.

One of the advantages of Ethereum Classic is that it is an original, very solid and valid project that is supported by many important investors. In fact, the ETC coin is still among the cryptocurrencies with the highest capitalization in the market. As a disadvantage, the blockchain strongly believes in the immutability of the code and does not have access to the latest updates and new features implemented by Ethereum (ETH)[11]. In recent years, it has also suffered from numerous network attacks the so called 51% attacks[12].

[11] Source: https://www.redditoinclusione.it/ethereum-classic/

[12] https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/

Figure 2.5. ETC Logo[13]

2.4 Blockchain Logging Framework (BLF)

Blockchain Logging Framework is an application written in Java that deals with extracting data from blockchains. It was developed by *TU-Berlin Advanced Distributed System Prototyping* and is a fork of *Ethereum-Logging-Framework* (ELF) [8], an application that enables the analysis of data from the Ethereum blockchain. Currently, BLF supports the Ethereum network and Hyperledger. Through the connection to a node on the blockchain, it enables the extraction of information and the production of log files such as XES and CSV. In order to use the application it is required to provide as input a manifest file (*bcql*) which includes, in addition to the connection to a node of the blockchain, the data to extract such as the blocks to analyze, the smart contract, the extraction method to perform and the output file type.

The application is composed of several elements:

- *BcQL*: query language used for data on blockchains
- *Validator*: component that inspects the bcql manifest for specification errors
- *Extractor*: component to extract, process and format data
- *Generator*: component used to generate powerful logging functionality that can be incorporated into smart contracts

[13]https://cryptologos.cc/ethereum-classic

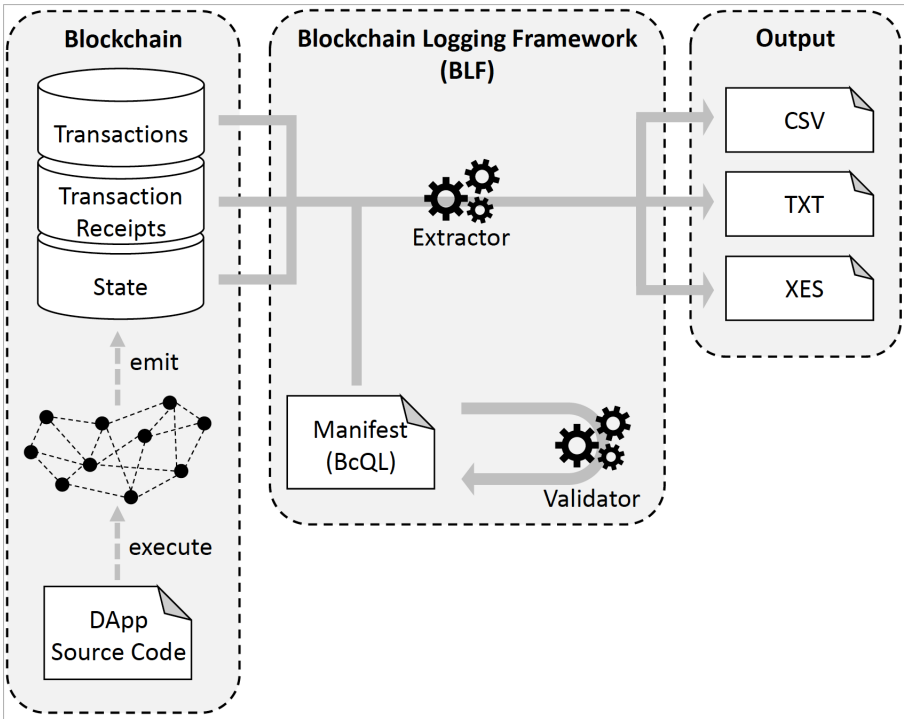


Figure 2.6. Summary of how BLF works[14]

More information regarding operation is available at the open source code of the application and on the papers [1, 9].

Another interesting application regarding the extraction of event logs from blockchain is *AlgoLogMiner* developed by Silverio Manganaro. It consists of a program written in Python that, through the Algorand Indexer, is able to generate event logs in XES format from the Algorand blockchain [11].

2.5 Process mining

Process mining represents a relatively new field of research that falls between data mining and process modeling and analysis. The main purpose of process mining is to understand, monitor and improve real processes through the extraction of log knowledge, available in large quantities in information systems.

Process mining deals mainly with: discovering the processes (*discovery*), which means generating a model of process thanks to an event log; checking the conformity (*conformance checking*), which consists in verifying the possible differences between a model and a log; discovering social networks (*social network*) and organizational networks; generating automatically simulation models; extending and re-evaluating models (*enhancement*); predicting future evolutions of a process model [5].

[14]https://github.com/TU-ADSP/Blockchain-Logging-Framework

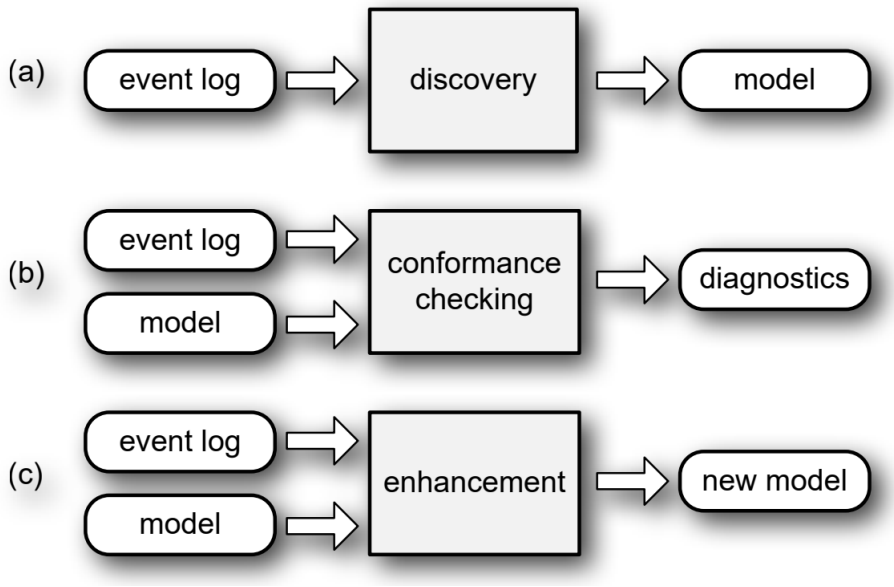


Figure 2.7. The three main use cases of process mining with input and output[15]

To carry out the process mining it is necessary to have an event log which should contain logs with at least a unique identification (*CaseID*), the *activity* (description of the event that is taking place) and the *timestamp*. It is possible do add also other information such as the *resource* and the *cost*.

The standard format used for process mining is *XES*. XES is based on XML and its task is to solve the problem of transferring the syntax and semantics of an event from the place of generation to the place of analysis, making it easily understood by both parties. In 2009, to promote research, development and education on this topic, *IEEE Task Force* on process mining was founded.

[15] https://www.tf-pm.org/upload/1580737614108.pdf

Chapter 3

Approach and Implementation

3.1 Approach

In this section we describe in general the approach used to implement the blockchain of Algorand and Ethereum Classic on the Blockchain Logging Framework application.

3.1.1 Algorand

An overview of the approach regarding the implementation of Algorand can be seen through this diagram:

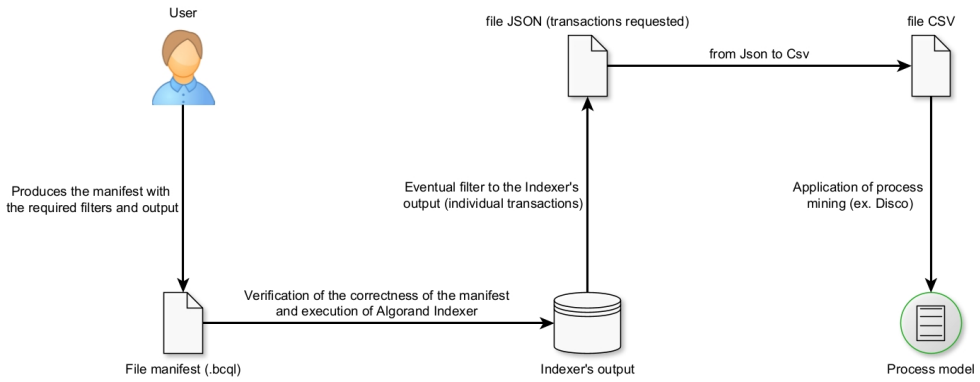


Figure 3.1. Diagram representing the approach for Algorand

User's task is to provide a manifest (*bcql*) file with the information on the data to extract, as is normally done on BLF. This file will then be checked to ensure that the data required for the extraction is present (such as the connection to the

Indexer) and that the information submitted is correct. Subsequently, a connection to an *Algorand REST API Indexer* occurs, a tool that simplifies the extraction of transactions from the Algorand blockchain, which takes the user's input, begins to search and produces the required data as output. This data is then formatted and, if required, filters are applied on individual transactions (such as the recipient). Once the required data has been retrieved and eventual filters have been applied to the individual transactions, a JSON file is generated with all the transactions found. Then, from that file, a CSV file is generated as output, which allows process mining. Regarding Algorand, it has been implemented the possibility to generate as output only CSV files, even if this should not be a problem since it is possible to generate XES files starting from CSV thanks to process mining programs like *Disco*[1].

3.1.2 Ethereum Classic

As far as the Ethereum Classic blockchain is concerned, implementation was almost straightforward. This is due to the fact that BLF already supports Ethereum blockchain. Since Ethereum was created from a hard fork of Ethereum Classic network they use the same programming language, *Solidity*, which is strongly supported by BLF. Therefore it was sufficient to add Ethereum Classic to the *BLF Validator* (which checks the correctness of the bcql manifest file) and use the same *Extractor* (component that extracts the required transactions) as Ethereum. In this way it is possible to use the same functions that already exist for Ethereum to search for data and generate both XES and CSV files as output.

3.2 Implementation

In this paragraph we take a closer look at the implementation of the two blockchains on Blockchain Logging Framework (BLF) and the various changes that have been made to the pre-existing files. It is possibile to view the open source code with the changes made to BLF on Github[2].

3.2.1 Manifest changes

In order to function, BLF requires a manifest file (*bcql*) from the user carrying out the search, into which the required data is entered.

Ethereum Classic

As far as the syntax of ETC's manifest is concerned, no changes have been made to its functioning, they are identical to those of Ethereum. For information on how it works and how to use it, it is possible to refer directly to the official project Github[3].

[1] https://fluxicon.com/disco/
[2] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework
[3] https://github.com/TU-ADSP/Blockchain-Logging-Framework

Algorand

Contrary to ETC, some changes have been made to the Algorand blockchain since the Indexer supports numerous transaction filters. The following image shows an example of a manifest file regarding Algorand.

```
SET BLOCKCHAIN "Algorand";
SET OUTPUT FOLDER "./test_output";
SET CONNECTION {"https://testnet-algorand.api.purestake.io/idx2", "qPBSYEwBfX7BYAjkNDSel8BCMF5HZJJo9NO2F2R2"};

BLOCKS () ()
{
    LOG ENTRIES ("RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKTVFXAXSU744OE") (payment-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "mine_total", CaseID "me", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKTVFXAXSU744OE") (payment-transaction)
    {
        TRANSACTION FILTERS (receiver JX3XXWRGLONZ6PLNQXHQ6DBCUELFXFWY35VGD5IVHALKJYWLJU2TFAII3A);
        EMIT CSV ROW (Activity "mine_jx3", CaseID "me", Timestamp round-time, Quantity amount);
    }
}
```

Figure 3.2. An example of a manifest file for the Algorand blockchain

In this simple example, the application, starting from the manifest, generates a CSV file containing all ALGO transactions made by a given user on the Algorand testnet (*RBR* address), and also filters out those that are addressed to a chosen address (*JX3*).

Data to include in the Algorand Manifest

As in the normal operation of BLF, blockchain is chosen at first, followed by the output directory and a connection to the Indexer, in this case via an API[4]. Afterwards, the blocks which have to be analysed are selected via the *BLOCKS* line. The input must be entered in the brackets; the first indicates from which block and the second up to which one. If no data is entered in the brackets, then all blocks in the blockchain are searched.

Next, inside the body of *BLOCKS* it is possible to choose the data to analyse through the line *LOG ENTRIES*. Each body of *LOG ENTRIES* indicates a call to the Indexer, so it is possible to make several calls with different filters (at least one). The first piece of data to add in *LOG ENTRIES* must be the address, the asset or the smart contract from which we want to extract the data, while the second must be the type of transaction. In our case it is a *payment-transaction* which indicates a simple ALGO transfer.

There are different transaction types such as *asset-transfer-transaction* which indicates an asset transfer and *application-transaction* which indicates a call to a smart contract. Depending on the type of transaction, it is possible to apply certain filters that will be executed by the Indexer. In our example there are no such filters, so all ALGO transactions from the address are extracted.

[4]https://www.purestake.com/

The applicable filters are:

- *asset_id* in the case in which we want to filter the transactions of a specific asset
- *address_role* through which we can specify the role of the address as sender, recipient or freeze-target
- *txn_type* that allows us to analyse the type of transaction
- *min_amount* if we need to filter transactions according to a minimum amount

It is possible to add support for additional filters by editing the *apply_Filters* file. A list of all applicable filters can be found on the Algorand developer portal[5]. Within the body *LOG ENTRIES* it is necessary also to have the line *TRANSACTION FILTERS* which adds the possibility of applying additional filters to individual transactions that are generated by the Indexer. In our example, no filters are applied in the first call, while in the second call, we filter out transactions whose recipient (*receiver*) is the address starting with *JX3*.

Lastly, the final line required is *EMIT CSV ROW* in which we enter the data we want to extract from the discovered transactions. At the very least we require *Activity* and *CaseID*. With Activity we indicate the task which we analyse and with CaseID the unique identification of the person who carried out the task. This data can be directly taken from the analysed transactions, or alternatively we have the possibility of inserting it through the use of apostrophes. Other examples of data that we want to extract, as in our case, can be *Quantity* that indicates the amount transferred or *Timestamp* that shows the exact execution time of the activity.

3.2.2 Algorand implementation

In this section we look in detail at how Algorand's blockchain was implemented on BLF.

BLF changes

As far as the BLF code is concerned, we modified the *BlfApp.java* file, which is the main file of the application and is responsible for starting the manifest verification and/or the data extraction. A check has been added to verify if the manifest concerns the Algorand blockchain.

[5] https://developer.algorand.org/docs/get-details/indexer/

```java
private static void extract(String filepath) throws NoSuchAlgorithmException, InterruptedException, IOException, ClassNotFoundException,
    NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {
    boolean Algo_true = false;
    boolean both = true;
    try {
        File check_type = new File(filepath);
        Scanner check_type2 = new Scanner(check_type);
        while (check_type2.hasNextLine()) {
            String data_check = check_type2.nextLine();
            if (data_check.contains("algorand") || data_check.contains("ALGORAND") || data_check.contains("ALGORAND")) {
                Algo_true = true;
                break;
            }
        }
        check_type2.close();
        if (Algo_true) {
            check_Manifest.startAlgo(filepath, both);
        } else {
            final Extractor extractor = new Extractor();
            try {
                extractor.extractData(filepath);
            } catch (BcqlProcessingException ex) {
                ex.printStackTrace(System.err);
            }
        }
```

Figure 3.3. Change made to the extract function of *BlfApp.java* file

If the blockchain is Algorand, the *startAlgo* function of the *check_Manifest.java* file is executed.

Check_Manifest.java file

This file, thanks to the *startAlgo* function, takes care of the manifest syntax and checks for the presence of all the data required to perform the extraction. Various checks are carried out with regard to:

- The output folder
- The Indexer connection
- The *BLOCKS* section
- The presence of at least one *LOG ENTRIES*
- The filters on single transactions
- The output

Afterwards, it takes care of creating copies of the *transactions_Finder0.txt* and *transactions_Finder1.txt* file depending on how many calls to the Indexer there are and execute the *implementFilters* function of the *apply_Filters.java* file for each generated file.

Since Java is a compiled language, it does not allow the execution of subsequently generated Java code. In order to execute such files and to solve this problem, we use *reflection*.

```
// Execute finder with reflection
File f = new File("target/classes");
URL[] cp = { f.toURI().toURL() };
URLClassLoader urlcl = new URLClassLoader(cp);
Class<?> clazz1 = urlcl.loadClass("blf.blockchains.algorand.transactions_Finder" + String.valueOf(v));
Method method = clazz1.getMethod(
    "finding",
    String.class,
    String.class,
    String.class,
    List.class,
    List.class,
    List.class,
    String.class,
    int.class,
    List.class
);
method.invoke(
    null,
    address,
    token,
    output_path,
    duplicates,
    duplicates_value,
    savings,
    String.valueOf(v),
    lung,
    saving_singles
);
urlcl.close();
```

Figure 3.4. Portion of code showing the use of reflection

As a last action, after generating the output CSV file, *startAlgo* also takes care of deleting all generated temporary files needed for the extraction of transactions.

Transactions_Finder0/1.txt file

These txt files are used to create copies of themselves in Java format. They contain the *finding* function which, via a link to an Indexer API, takes care of finding and extracting transactions from the Algorand blockchain and filtering the required data into output.

The first call to the Indexer generates the JSON file that will contain the transactions extracted, while the last one calls the *createCsv* function of the *json_To_Csv.java* file. Each single extracted transaction is, if required, filtered through the *single_Transaction.java* file.

Connection_To_Indexer.java file

File which, via an API with token and password, allows the connection to the Algorand Indexer.

Json_To_Csv.java file

It takes care of generating the ouput CSV file. It takes as input a JSON file and generates its equivalent in CSV[6]. It is called on the last execution of the Indexer.

[6] https://www.baeldung.com/java-converting-json-to-csv

If the JSON file is empty the CSV file is not generated. The implementation was made possible thanks to the library *jackson-dataformats-text*[7].

Single_Transaction.java file

Its purpose is to apply filters on individual transactions retrieved from the Indexer. An example of this is the recipient of the transaction or when the transaction was executed (timestamp).

Apply_Filters.java file

The task of *implementFilters* function of *apply_Filters.java* is to implement, for each file previously generated by the *startAlgo* function, the filters chosen by the user. This has been made possible thanks to several comments that are modified inside the files generated by *check_Manifest.java*.

```
while (numtx > 0)
{
    String next_page = nexttoken;
    String response = indexerClientInstance.searchForTransactions()
        .limit(limit)
        .next(next_page)

        // ValueToChange0
        // ValueToChangeMIN
        // ValueToChangeMAX
        // ValueToChange1
        // ValueToChange2
        // ValueToChange3
        // ValueToChange4
        // ValueToChange5
        // ValueToChange6
        // ValueToChange7
        // ValueToChange8

        .execute()
        .toString();
    JSONObject jsonObj = new JSONObject(response.toString());
    JSONArray jsonArray = (JSONArray) jsonObj.get("transactions");
    numtx = jsonArray.length();
```

Figure 3.5. Comments that are edited to apply filters to the Indexer calls

ValueToChange0 is changed with the address, smart contract or asset to analyse. *ValueToChangeMIN* with the starting block and *ValueToChangeMAX* with the final block to scan. The *ValueToChanges* ranging from 1 to 8 are optional and are only modified if the user applies filters to the Indexer.

3.2.3 Ethereum Classic implementation

In this section we take a look at how the Ethereum Classic blockchain was implemented on BLF.

[7]https://github.com/FasterXML/jackson-dataformats-text/tree/master/csv

BLF changes

The only change that has been made relates directly to the BLF code.

```
public class Constants {

    public static final String ETHEREUM_BLOCKCHAIN_KEY = "ethereum";
    public static final String ETHEREUM_CLASSIC_BLOCKCHAIN_KEY = "ethereum classic";
    public static final String HYPERLEDGER_BLOCKCHAIN_KEY = "hyperledger";
    public static final String TEMPLATE_BLOCKCHAIN_KEY = "template";

    private static final Map<String, BaseBlockchainListener> blockchainMap = new HashMap<>();

    public static Map<String, BaseBlockchainListener> getBlockchainMap(VariableExistenceListener variableExistenceListener) {
        blockchainMap.put(ETHEREUM_BLOCKCHAIN_KEY, new EthereumListener(variableExistenceListener));
        blockchainMap.put(ETHEREUM_CLASSIC_BLOCKCHAIN_KEY, new EthereumListener(variableExistenceListener));
        blockchainMap.put(HYPERLEDGER_BLOCKCHAIN_KEY, new HyperledgerListener(variableExistenceListener));
        blockchainMap.put(TEMPLATE_BLOCKCHAIN_KEY, new TemplateListener(variableExistenceListener));

        return blockchainMap;
    }
}
```

Figure 3.6. Change made to implement ETC

All it took to make ETC work was to add the constant "*ethereum classic*" to the *Costants.java* file, which is used by the *Validator* (which verifies the correct implementation of the manifest file) to understand whether the blockchain is supported and by the *Extractor* to know how to extract the data. It uses the same *Listener* as Ethereum since they use the same language (*Solidity*). The Listener extends the *BaseListener*, contains several useful functions for the Ethereum blockchain and takes care of initializing the data extraction through a stack.

Chapter 4

Validation

In this chapter we analyse several case studies to verify the correct behaviour of our implementation. Each example has been produced to test our modifications to BLF and concerns Algorand and Ethereum Classic blockchains.

First, we produced the bcql manifest with the data to extract. This file is then submitted to BLF. Regarding the extraction from the Algorand blockchain, the data is retrieved thanks to the link to an API of the Indexer[1]. In the case of the Ethereum Classic blockchain, the data is extracted through an API that allows direct connection to a network node[2]. Subsequently, BLF generates the output file which in the case of Algorand is a CSV file, while for Ethereum Classic is a CSV or XES file.

Finally, to visualise and analyse the event logs produced, we perform process mining. There are several applications that deal with process mining, examples of which are: ProM, Futura Reflect and Interstage Automated Process Discovery. In our case we will use the application Disco[3].

Figure 4.1. Disco logo[4]

Disco's peculiarities are: the production of excellent graphs, the display of the progress according to the timestamp, the generation of statistics, the possibility to filter the logs and the production of different types of output, even in XES format [7].

In total we will cover 4 cases on different topics:

[1]https://www.purestake.com/
[2]https://rivet.cloud/
[3]https://fluxicon.com/disco/
[4]https://processmi.com/programms/disco/

- The first concerns simple ALGO transactions
- The second concerning asset transfers on Algorand
- The third relates to smart contracts on Algorand
- The last one about an NFT project on the Ethereum Classic blockchain

The case studies analysed in this chapter can be consulted directly on Github[5].

4.1 Simple scenario

As a first case, we analyse a simple scenario on the Algorand blockchain, to be more precise on the testnet. *Testnet* denotes a blockchain identical to the original one, called the *mainnet*, which is mainly used to test smart contracts or to verify the correct implementation of updates on the blockchain before running them on the main network. For this reason, the exchange currency is worthless and can be redeemed by anyone for free, the so-called *faucet*[6].

Consequently, I created a wallet, redeemed the faucet and made several ALGO transactions to addresses randomly chosen from the explorer. The *explorer* is a tool that is present in all public blockchains and has, as one of its functionalities, the possibility to view all the transactions carried out on the network. The address that I generated and used for the test on the testnet is:
"*RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKT VFXAXSU744OE*".

An explorer can also be used to view and consult the transactions carried out for this case study. One of the best-known, as far as Algorand is concerned, is *AlgoExplorer*[7].

Figure 4.2. AlgoExplorer logo[8]

So in this use case, we are going to examine simple ALGO currency transactions.

[5]https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework
[6]https://bank.testnet.algorand.network/
[7]https://algoexplorer.io/
[8]https://developer.algorand.org/ecosystem-projects/

```
SET BLOCKCHAIN "Algorand";
SET OUTPUT FOLDER "./test_output";
SET CONNECTION {"https://testnet-algorand.api.purestake.io/idx2", "qPBSYEwBfX7BYAjkNDSe18BCMF5HZJJo9NO2F2R2"};

BLOCKS () ()
{
    LOG ENTRIES ("RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKTVFXAXSU744OE") (payment-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "mine_total", CaseID "me", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKTVFXAXSU744OE") (payment-transaction)
    {
        TRANSACTION FILTERS (receiver JX3XXWRGLONZ6PLNQXHQ6DBCUELFXFWY35VGD5IVHALKJYWLJU2TFAII3A);
        EMIT CSV ROW (Activity "mine_jx3", CaseID "me", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("RBRBZ2QOWDFM7WTGXGBME72KANAJZ6OCUR4X6LTLQ53PKTVFXAXSU744OE") (payment-transaction)
    {
        TRANSACTION FILTERS (receiver EX3L3OW2EOCHDJCUX3XDGDIQQZFVNOSSRVCG4ZK73VLZNZ2EO4ZBT2HE6U);
        EMIT CSV ROW (Activity "mine_ex3", CaseID "me", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("RYRYZOAQNG6SAD3XQMRIRH3VR7TRWIMYR5BZLA4OEKUEQXI7ZO62I74BQM") (payment-transaction, txn_type PAY)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "random1_total", CaseID "random1", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("QIFS3OCHBM5UPOPNWJS4BE6PSNJ6O5RYDMXB7I4HX6Y226CLPQ2KBT3SHY") (payment-transaction, txn_type PAY)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "random2_total", CaseID "random2", Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("44SDV4IGZWGQFSEQW2NONO3YVQZGT4KLSX53XZS7JHDQSFGONDH6IOPS2U") (payment-transaction, txn_type PAY)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "random3_total", CaseID "random3", Timestamp round-time, Quantity amount);
    }
}
```

Figure 4.3. Manifest of the first case study[9]

As can be seen from the manifest, we analyse all the blocks of the network. The extraction was completed in about 42 seconds. In total there are 6 calls to the Indexer:

- The first one concerns my own address and we extract transactions of type *payment-transaction* which indicates ALGO sendings
- The second and third concern still my transactions but the results are filtered. In the first case they relate to the recipient of the address with the initials *JX3* and in the second the one with initials *EX3*
- In the fourth call we extract all ALGO transactions made by the user with the initials *RYR*
- In the fifth call all ALGO transactions made by the user *QIF*
- In the sixth the transactions of the user *44S*

In the last three calls we apply the *txn_type PAY* filter since the chosen addresses perform different types of transactions, while we only want normal transfers.

[9]https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/1%20-%20MIO%20ALGORAND%20(TESTNET)/manifest1.txt

The output is a JSON file which in turn generates the CSV. A portion of the CSV file is visible in the image 4.4. As can be seen, it consists of several columns:

- *Activity*, which shows the action performed
- *CaseID*, that indicates the user who carried out the transaction
- *Timestamp*, used to sort the transactions according to the time of execution
- *Quantity*, which represents the number of ALGOs sent (in microAlgo)

```
Activity,Quantity,CaseID,Timestamp
mine_total,1000000,me,"2021-12-07 04:28:40.000"
mine_total,800000,me,"2021-12-07 00:54:54.000"
mine_total,700000,me,"2021-12-07 00:54:46.000"
mine_total,100000,me,"2021-12-07 00:54:37.000"
mine_total,200000,me,"2021-12-07 00:54:29.000"
mine_total,400000,me,"2021-12-07 00:54:24.000"
mine_total,400000,me,"2021-12-07 00:54:16.000"
mine_total,300000,me,"2021-12-07 00:54:12.000"
mine_total,200000,me,"2021-12-07 00:54:03.000"
mine_total,100000,me,"2021-12-07 00:53:59.000"
mine_total,100000,me,"2021-12-07 00:53:46.000"
mine_total,100000,me,"2021-12-07 00:53:38.000"
mine_ex3,400000,me,"2021-12-07 00:54:24.000"
mine_ex3,100000,me,"2021-12-07 00:53:38.000"
mine_ex3,100000,me,"2021-12-07 00:53:34.000"
mine_ex3,100000,me,"2021-12-07 00:53:00.000"
mine_ex3,100000,me,"2021-12-07 00:52:51.000"
mine_ex3,100000,me,"2021-12-07 00:52:22.000"
mine_ex3,100000,me,"2021-12-07 00:51:31.000"
mine_ex3,100000,me,"2021-12-07 00:51:18.000"
random1_total,10000000,random1,"2021-03-24 09:13:41.000"
random2_total,2000,random2,"2022-01-21 01:08:10.000"
```

Figure 4.4. Example of the CSV produced[10]

This CSV file is then passed to the Disco application for process mining.

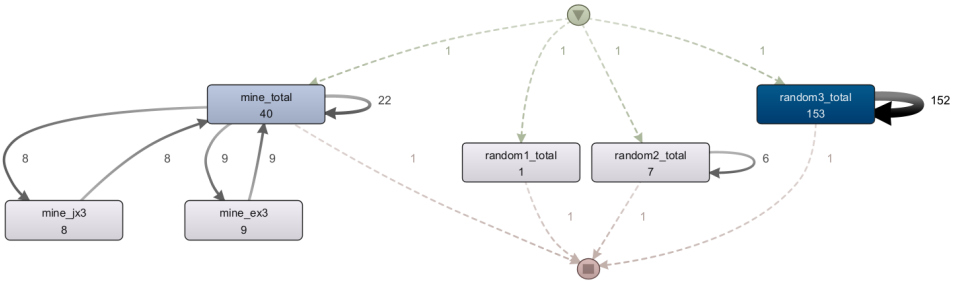


Figure 4.5. Transaction graph produced by Disco

[10]https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/1%20-%20MIO%20ALGORAND%20(TESTNET)/output.csv

As can be seen from the graph 4.5, the more the colour tends towards blue, the more transactions the user has performed. As for my address, I performed 40 transactions, 8 to *JX3* and 9 to *EX3*. The other addresses performed 1, 7 and 153 transactions.

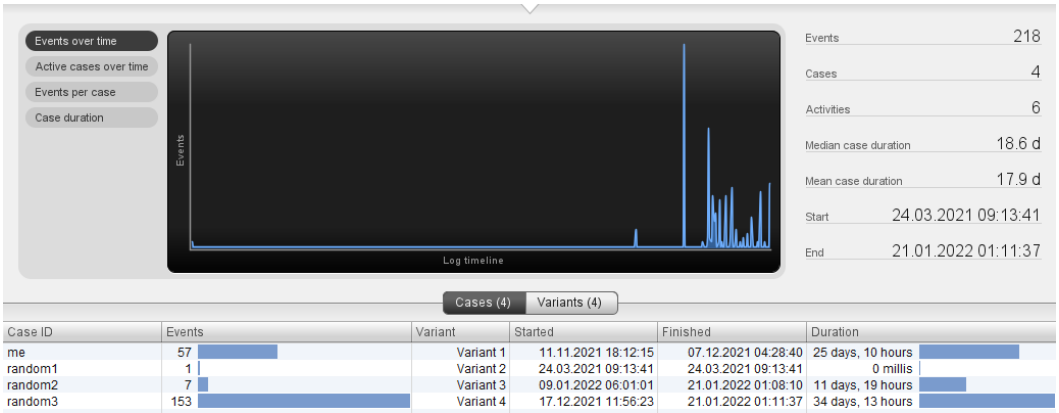


Figure 4.6. Statistics of analysed event logs

Disco, as previously stated, also generates a lot of statistical data. An example is the image 4.6 where we can see the total number of events, in this case 218, the average duration of 18 days, the start and end of the execution.

4.2 Alchemon DApp

The second case analysed concerns the decentralised application *Alchemon* developed on Algorand's blockchain[11]. Alchemon is one of the first trading card games on the Algorand network. It consists of trading, evolving, crafting and collecting NFTs.

Figure 4.7. Alchemon logo[12]

The case that we are going to analyse concerns *AlcheCoin*, the ecosystem asset (token). AlcheCoin at this moment has no real function, but in the future it might be used for different purposes. However, there is still the possibility of staking the token and receiving AlcheCoin passively every week. This is possible by adding a second asset with id *320570576* to your own address.

[11] https://alchemon.net/
[12] https://www.algopulse.io/project-landing

Therefore we will analyze through the event logs the transfer of these two assets in order to track the number of transactions and the number of users that staked the token.

```
SET BLOCKCHAIN "Algorand";
SET OUTPUT FOLDER "./test_output";
SET CONNECTION {"https://mainnet-algorand.api.purestake.io/idx2", "qPBSYEwBfX7BYAjkNDSel8BCMF5HZJJo9NO2F2R2"};

BLOCKS (18720439) (18739639)
{
    LOG ENTRIES ("310014962") (asset-transfer-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "transfer", CaseID sender, Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("320570576") (asset-transfer-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "staking", CaseID sender, Timestamp round-time, Quantity amount);
    }
}
```

Figure 4.8. AlcheCoin token manifest[13]

As can be seen from the image 4.8, we analyse transactions from block *18720439* to *18739639*, which corresponds to approximately 23 hours. The extraction process lasted about 16 seconds.

The Indexer performs two searches:

- The first one for the asset with id *310014962* denoting AlcheCoin
- The second for the asset with id *320570576* which indicates the flag needed in order to stake AlcheCoin

For both searches, we specify *asset-transfer-transaction* as the transaction type.

As output we have:

- *Activity*: transfer or staking
- *CaseID*: the user who performed the action
- *Timestamp*
- Amount transferred

In output we obtain a CSV file[14] that we transfer to Disco.

[13] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/2%20-%20ALCHEMON%20ALGORAND/manifest.txt

[14] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/2%20-%20ALCHEMON%20ALGORAND/output.csv

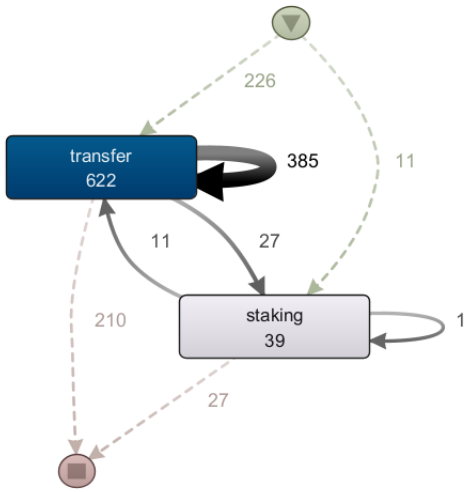


Figure 4.9. AlcheCoin token transaction graph

From the graph 4.9, we can see that there were 622 transfers and 39 users staked their tokens. There were also 11 transactions after staking, 27 staked after a transaction, many users did multiple transfers (385) and one user enabled the staking flag twice in a row.

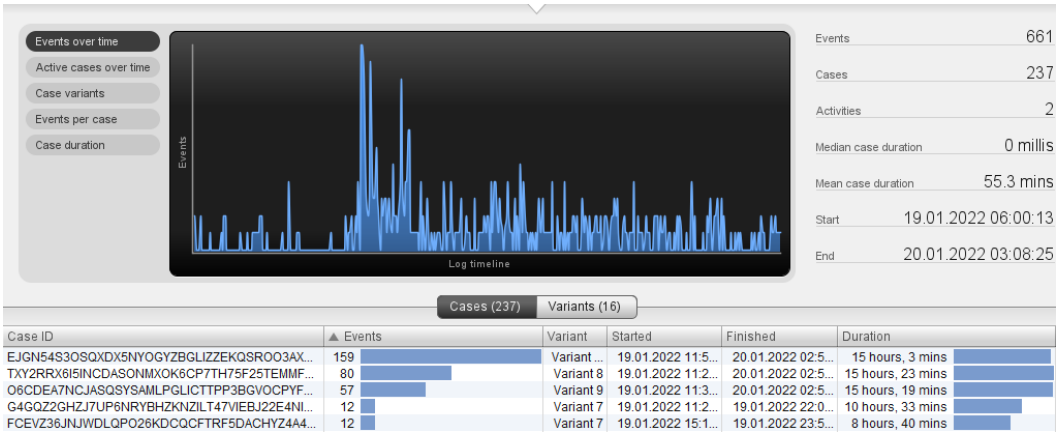


Figure 4.10. AlcheCoin token statistics

From the statistics in the figure 4.10 we can see that there have been 661 events from 237 addresses and the average execution time is 55 minutes. The data also shows us a peak of events occurring at 12:04 on 19-01-2022. The address with the most activity was the one starting with *EJG* with a total of 159 events.

4.3 Yieldly.finance DApp

The last analysis carried out on the Algorand blockchain concerns the *Yieldly.finance* DApp[15].

Figure 4.11. Yieldly.finance logo[16]

It is one of the first DApps on DeFi on Algorand and defines itself as the application that offers the highest rewards. Its main functionality is to allow users to take advantage of financial instruments in a decentralised way.

It can be used for:

- Liquidity mining
- Participate in the no-lose lottery
- Stake of YLDY asset
- Swap cross-chain

Liquidity mining refers to the ability to deposit liquidity (ALGO or Algorand assets) to DApps and receive rewards in return.
Cross-chain swap means the possibility to exchange assets between different blockchains.

The case study of this DApp concerns the main smart contracts of the ecosystem that can be found on the official website[17].

[15] https://yieldly.finance/
[16] https://yieldly.finance/
[17] https://yieldly.finance/security/

The main contracts are:

- Staking
- Lottery
- ALGO escrow
- YLDY asset escrow
- Proxy

The *escrow* is a smart contract that is called before or after a smart contract transaction is carried out. Its task is to hold ALGO or the assets until the transfer conditions are met[18].

As for the *proxy* smart contract, since the smart contracts written on the blockchain are in most cases immutable, if an error is discovered, there is no possibility of correcting it. This is why the proxy is used. Its job is to store the latest version of the called smart contract and to redirect calls to that currently valid logic. So if the contract logic is updated and a new smart contract is deployed, it is only necessary to update the proxy contract reference variable[19].

```
SET BLOCKCHAIN "Algorand";
SET OUTPUT FOLDER "./test_output";
SET CONNECTION {"https://mainnet-algorand.api.purestake.io/idx2", "qPBSYEwBfX7BYAjkNDSel8BCMF5HZJJo9NO2F2R2"};

BLOCKS (18735217) (18739217)
{
    LOG ENTRIES ("233725848") (application-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "proxy", CaseID sender, Timestamp round-time, Quantity "0");
    }

    LOG ENTRIES ("233725844") (application-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "lottery", CaseID sender, Timestamp round-time, Quantity "0");
    }

    LOG ENTRIES ("233725850") (application-transaction)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "staking", CaseID sender, Timestamp round-time, Quantity "0");
    }

    LOG ENTRIES ("FMBXOFAQCSAD4UWU4Q7IX5AV4FRV6AKURJQYGXLW3CTPTQ7XBX6MALMSPY") (asset-transfer-transaction, address_role sender)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "escrow_to_user_yldy", CaseID receiver, Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("FMBXOFAQCSAD4UWU4Q7IX5AV4FRV6AKURJQYGXLW3CTPTQ7XBX6MALMSPY") (payment-transaction, address_role sender)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "escrow_to_user_algo", CaseID receiver, Timestamp round-time, Quantity amount);
    }
    LOG ENTRIES ("FMBXOFAQCSAD4UWU4Q7IX5AV4FRV6AKURJQYGXLW3CTPTQ7XBX6MALMSPY") (payment-transaction, address_role receiver)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "user_to_escrow_algo", CaseID sender, Timestamp round-time, Quantity amount);
    }

    LOG ENTRIES ("FMBXOFAQCSAD4UWU4Q7IX5AV4FRV6AKURJQYGXLW3CTPTQ7XBX6MALMSPY") (asset-transfer-transaction, address_role receiver)
    {
        TRANSACTION FILTERS ();
        EMIT CSV ROW (Activity "user_to_escrow_yldy", CaseID sender, Timestamp round-time, Quantity amount);
    }
}
```

Figure 4.12. Yieldly.finance manifest[20]

[18] https://research.csiro.au/blockchainpatterns/general-patterns/blockchain-payment-patterns/escrow-2/

[19] https://blog.openzeppelin.com/proxy-patterns/

In this case study, we analyse the blocks from *18735217* to *18739217*, which are equivalent to approximately 5 hours. The extraction took about 1 minute and 40 seconds.

Seven calls are made to the Indexer:

- The first concerns proxy calls
- The second lottery calls
- The third staking calls
- In the fourth, we analyse all the transactions of the YLDY asset which carries out the escrow address (*FMB* initials) to the users
- In the fifth, we analyse all ALGO transactions made by the escrow address to the users
- In the sixth, we set the escrow address as the receiver and filter all ALGO transactions that users made to that address
- In the seventh, all transactions of the YLDY asset made by users towards the escrow

In output is then generated:

- The activity carried out
- Who held the action
- The timestamp
- The amount transferred, as regards to smart contract calls, it was manually set to 0

[20] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/3%20-%20YIELDLY.FINANCE%20ALGORAND/manifest.txt

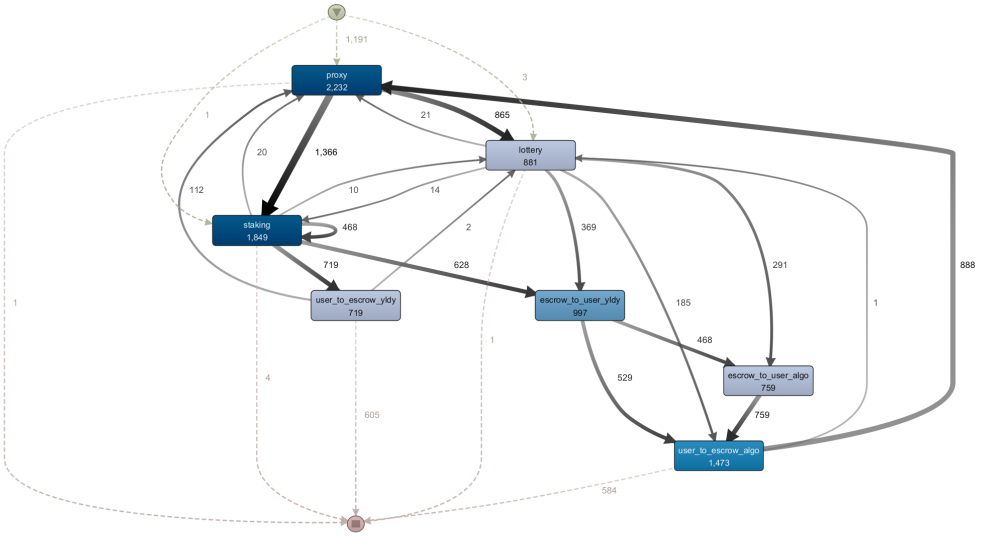


Figure 4.13. Graph of smart contract calls on Yieldly.finance

The graph 4.13 shows the process model of the event logs generated by Disco from the CSV file[21]. The presence of numerous variants makes it difficult to understand the execution. In order to try to understand the model better, it is necessary to analyse the statistics.

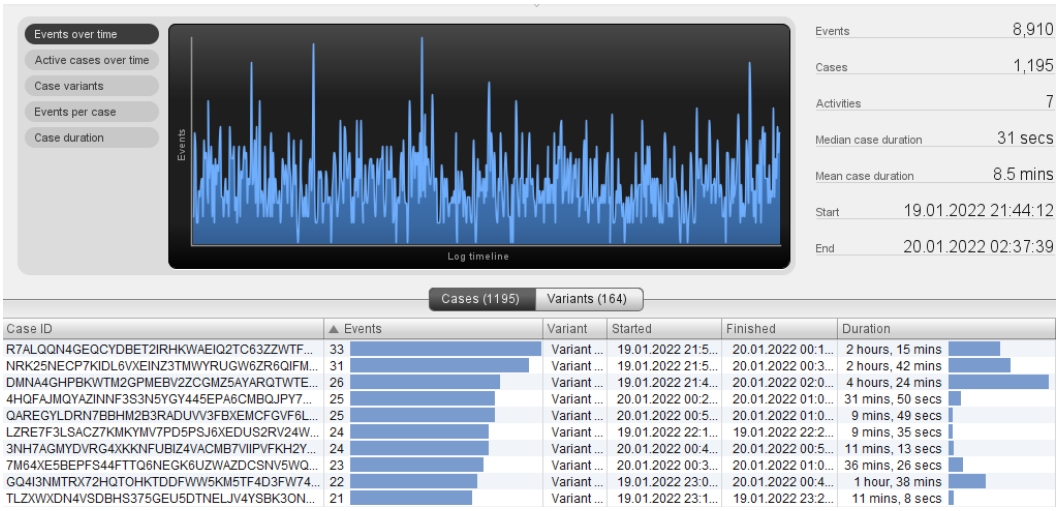


Figure 4.14. Yieldly.finance statistics

From the figure 4.14 we see that during these 5 hours there were 8910 events performed by 1195 users and the average duration was 31 seconds. The address that carried out the most transactions was the address starting with *R7A* with 33 events.

[21] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/3%20-%20YIELDLY.FINANCE%20ALGORAND/output.csv

In order to deepen and understand the functioning of Yieldly.finance, as we said before, we need to analyse the statistics produced by Disco. What we could establish is that there are several cases, let's analyse them.

4.3.1 YLDY asset staking calls

There are several cases concerning the staking of YLDY, which we will analyse thanks to different images.

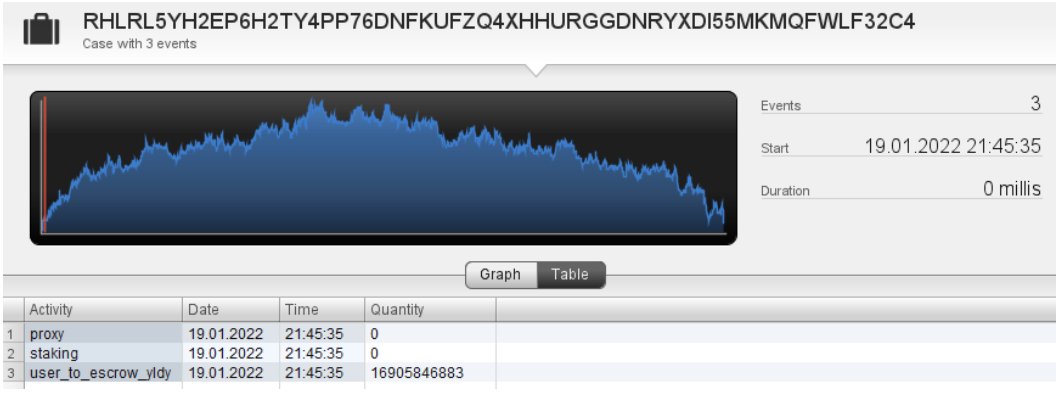


Figure 4.15. Action carried out when a user stacks the YLDY asset

From the image 4.15 we can observe the user with address *RHL* who puts in stake his YLDY asset. As we can see, the staking call performs 3 actions: the first action is the call to the proxy, which in turn makes the call to the right smart contract for staking. The user then sends his YLDYs to the escrow address which will hold them until the user wants to stake. This is done all in the same block.

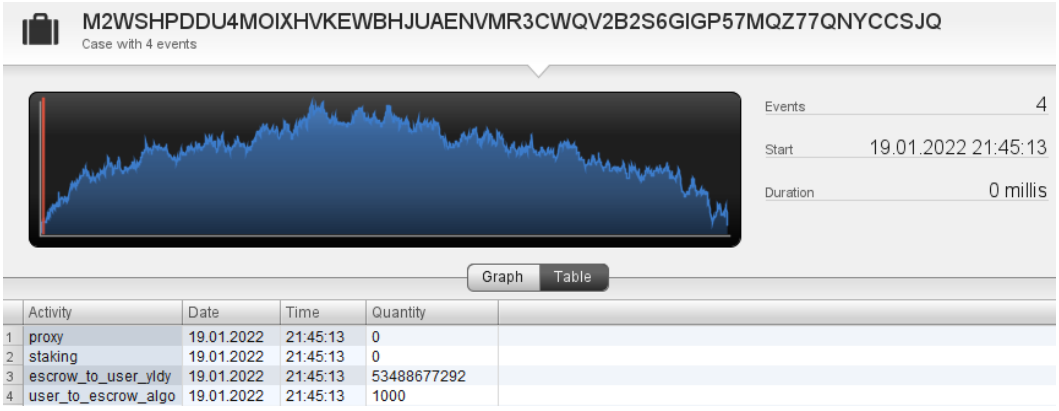


Figure 4.16. Action that takes place when a user withdraws the asset YLDY from staking

The image 4.16 instead shows the action that is performed when the user removes the asset YLDY from the staking. Four calls are made: as always at the beginning we have the call to the proxy which in turn calls the staking smart contract. Then the escrow address takes care of sending the right amount to the user. Finally the user sends a confirmation transaction to the escrow.

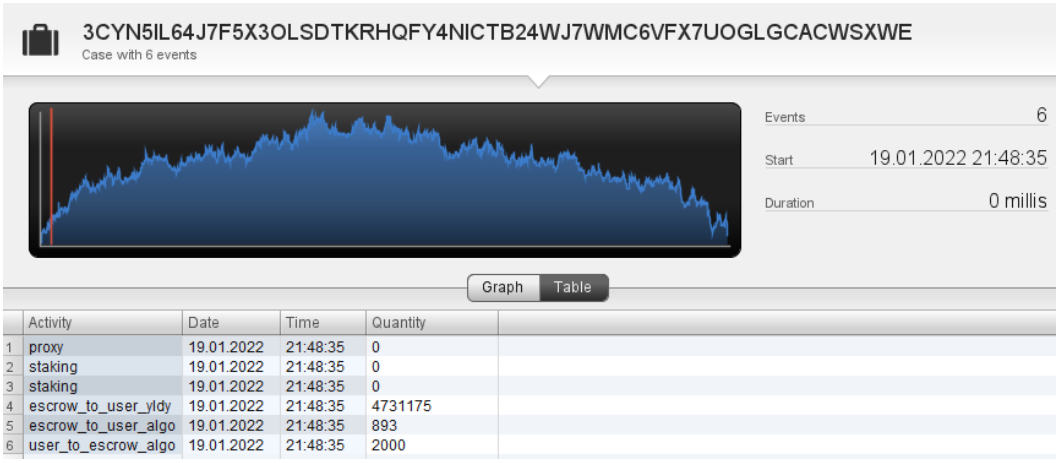


Figure 4.17. Action carried out when a user withdraws the asset YLDY and ALGO from staking

As far as the image 4.17 is concerned, it still involves the user who removes YLDY from the staking, but in this case there are two calls to the staking smart contract, as the user not only receives YLDY but also withdraws ALGO.

4.3.2 Lottery calls

Regarding the lottery smart contract, we have several actions here as well.

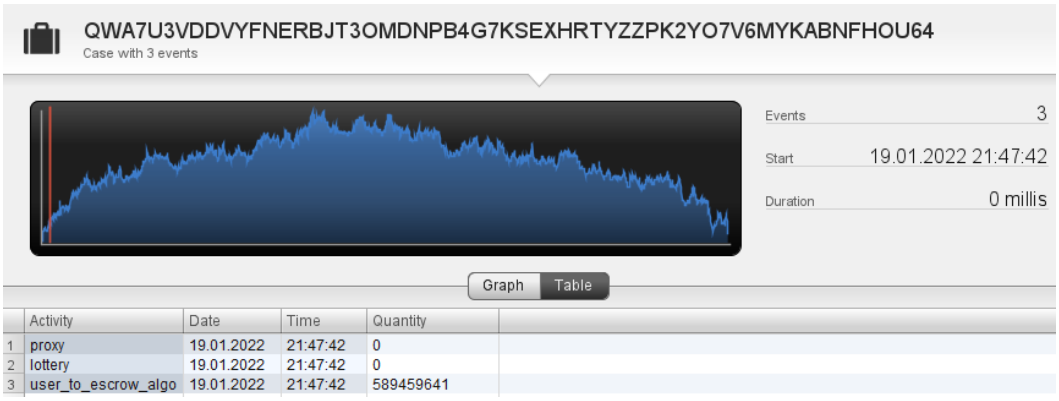


Figure 4.18. Action performed when a user wants to participate in the lottery

The image 4.18 shows the actions that are performed when the user participates in the lottery: first the proxy is called which redirects to the right smart contract of the lottery and then the user sends his ALGO to the escrow.

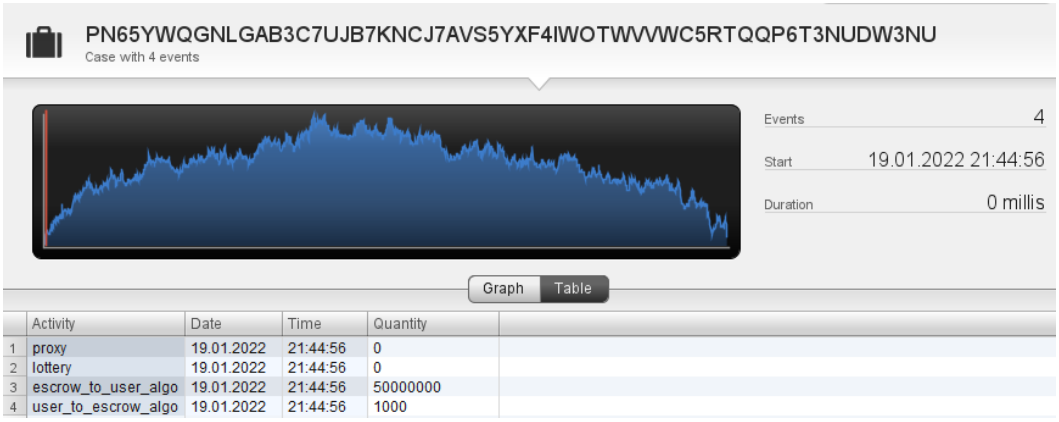


Figure 4.19. Action carried out when a user collects ALGO lottery rewards

The image 4.19 shows the actions that are performed when the user collects the ALGO rewards won from the lottery. There are 4 actions: the proxy, the call to the lottery, the escrow that sends the prize to the user and finally the user confirming through a transaction.

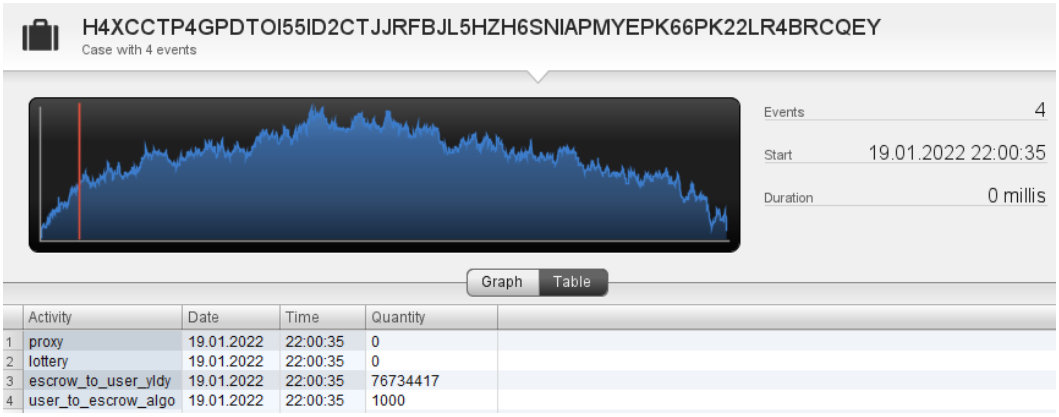
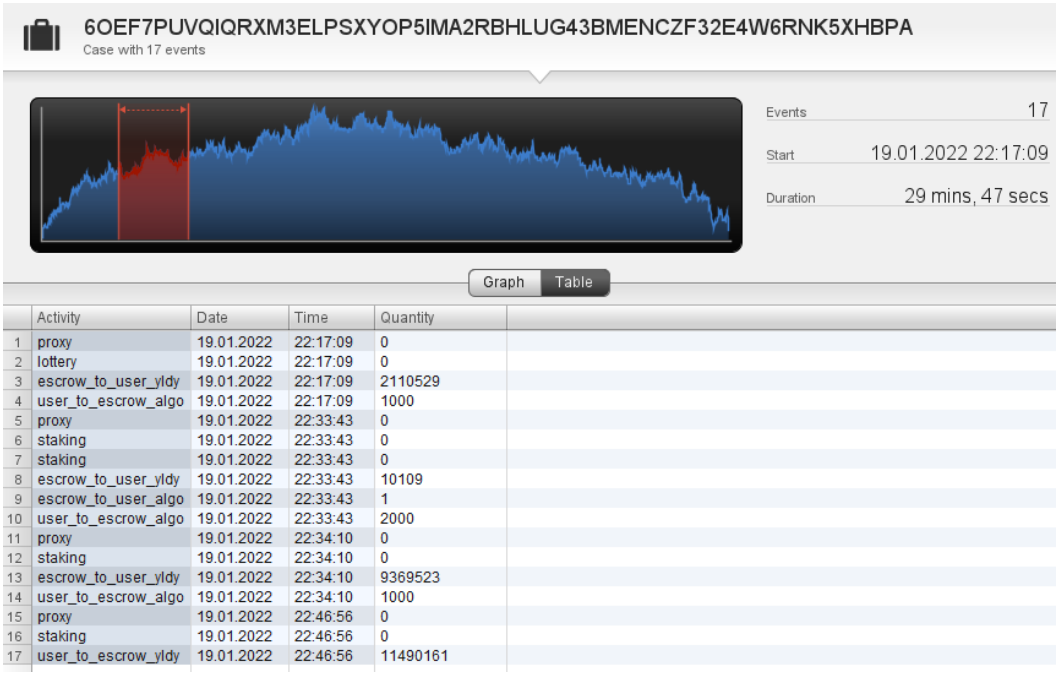


Figure 4.20. Action that occurs when a user collects rewards in YLDY from the lottery

The image 4.20 is also about the user withdrawing his winnings, but in this case it concerns the asset YLDY.



6OEF7PUVQIQRXM3ELPSXYOP5IMA2RBHLUG43BMENCZF32E4W6RNK5XHBPA
Case with 17 events

Events 17
Start 19.01.2022 22:17:09
Duration 29 mins, 47 secs

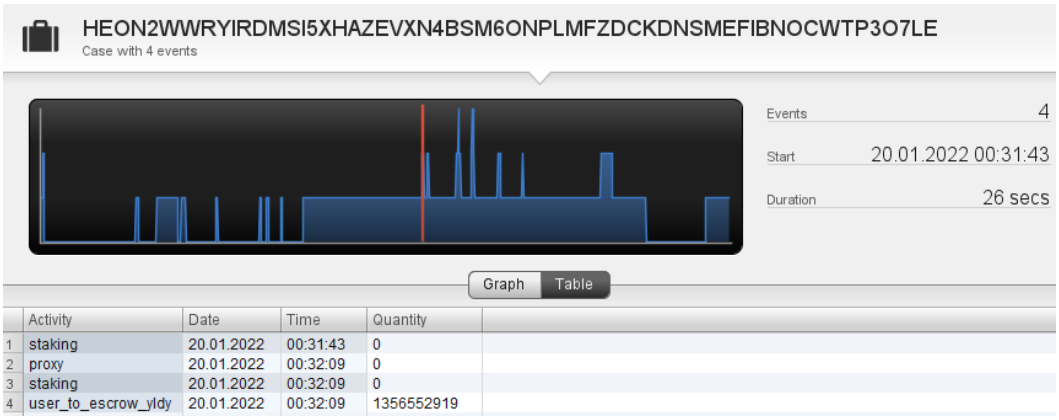
Graph Table

	Activity	Date	Time	Quantity
1	proxy	19.01.2022	22:17:09	0
2	lottery	19.01.2022	22:17:09	0
3	escrow_to_user_yldy	19.01.2022	22:17:09	2110529
4	user_to_escrow_algo	19.01.2022	22:17:09	1000
5	proxy	19.01.2022	22:33:43	0
6	staking	19.01.2022	22:33:43	0
7	staking	19.01.2022	22:33:43	0
8	escrow_to_user_yldy	19.01.2022	22:33:43	10109
9	escrow_to_user_algo	19.01.2022	22:33:43	1
10	user_to_escrow_algo	19.01.2022	22:33:43	2000
11	proxy	19.01.2022	22:34:10	0
12	staking	19.01.2022	22:34:10	0
13	escrow_to_user_yldy	19.01.2022	22:34:10	9369523
14	user_to_escrow_algo	19.01.2022	22:34:10	1000
15	proxy	19.01.2022	22:46:56	0
16	staking	19.01.2022	22:46:56	0
17	user_to_escrow_yldy	19.01.2022	22:46:56	11490161

Figure 4.21. Example of transactions carried out by a user

Finally, in the image 4.21, we analyse the execution of several calls made by the user with the initials *6OE*. From the cases identified above we can see that the user has: as a first withdrew the YLDY rewards from the lottery, then unstacked YLDY and ALGO, afterwards unstacked YLDY again and finally staked YLDY. He performed these actions in approximately 30 minutes.

Thanks to Disco we were also able to find anomalous cases. An example is the picture 4.22 where we can see that the user with the initials *HEO* called the staking smart contract directly without going through the proxy first.



HEON2WWRYIRDMSI5XHAZEVXN4BSM6ONPLMFZDCKDNSMEFIBNOCWTP3O7LE
Case with 4 events

Events 4
Start 20.01.2022 00:31:43
Duration 26 secs

Graph Table

	Activity	Date	Time	Quantity
1	staking	20.01.2022	00:31:43	0
2	proxy	20.01.2022	00:32:09	0
3	staking	20.01.2022	00:32:09	0
4	user_to_escrow_yldy	20.01.2022	00:32:09	1356552919

Figure 4.22. Example of an unusual case

4.4 ETCBayc DApp

As regards the Ethereum Classic blockchain, we analyse the *ETCBayc* application[22].

Figure 4.23. ETCBayc logo[23]

ETCBayc is an NFT project developed on ETC after the great success of *Bored Ape Yacht Club* on Ethereum. The project consists of 10,000 unique NFTs to collect. The NFTs represent chimpanzees with different characteristics and outfits. In this analysis we will look at the execution of several smart contract functions inherent to these NFTs.

```
SET BLOCKCHAIN "Ethereum Classic";
SET OUTPUT FOLDER "./test_output";
SET CONNECTION "wss://569baae4d6ef4178a548f1835d3c9294.etc.ws.rivet.cloud/";

BLOCKS (14304089) (14310089) {

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (Mint(uint indexed tokenId, address indexed mintedBy, address indexed mintedTo)) {
        EMIT XES EVENT ()(tokenId)()("mint" as xs:string concept:name);
    }

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (TokenOnSale(uint256 indexed _tokenId, address indexed _owner, uint256 _price)) {
        EMIT XES EVENT ()(_tokenId)()("onSale" as xs:string concept:name);
    }

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (TokenNotOnSale(uint256 indexed _tokenId, address indexed _owner)) {
        EMIT XES EVENT ()(_tokenId)()("notOnSale" as xs:string concept:name);
    }

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (MarketTrade(uint256 indexed _tokenId, address indexed _from, address indexed _to, address buyer, uint256 _price)) {
        EMIT XES EVENT ()(_tokenId)()("trade" as xs:string concept:name);
    }

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (TokenMarketPriceChange(uint256 indexed _tokenId, address indexed _owner, uint256 _oldPrice, uint256 _newPrice)) {
        EMIT XES EVENT ()(_tokenId)()("priceChange" as xs:string concept:name);
    }

    LOG ENTRIES (0x59E34EF31049565D041Aec6137F40f518c2D47c1) (Transfer(address indexed from, address indexed to, uint256 indexed tokenId)) {
        EMIT XES EVENT ()(tokenId)()("transfer" as xs:string concept:name);
    }

}
```

Figure 4.24. ETCBayc Manifest[24]

Regarding the manifest 4.24, we analyse blocks from *14304089* (10 January 2022 at 10:23 CET) to *14310089* (11 January 2022 at 8:11 CET). The data were extracted in about 43 minutes.

The analysed functionalities are:

- *Mint*, NFT generation from smart contract

[22] https://etcbayc.com/
[23] https://twitter.com/etcbayc/photo
[24] https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/4%20-%20ETCBAYC%20ETC/manifest4.txt

- *TokenOnSale*, offer for sale of an NFT on marketplace
- *TokenNotOnSale*, removal of an NFT sale from the marketplace
- *MarketTrade*, sale of an NFT
- *TokenMarketPriceChange*, change of the sale price of an NFT
- *Transfer*, NFT transfer

A XES file is generated as output with the events as activity and the NFT in question as CaseID[25].

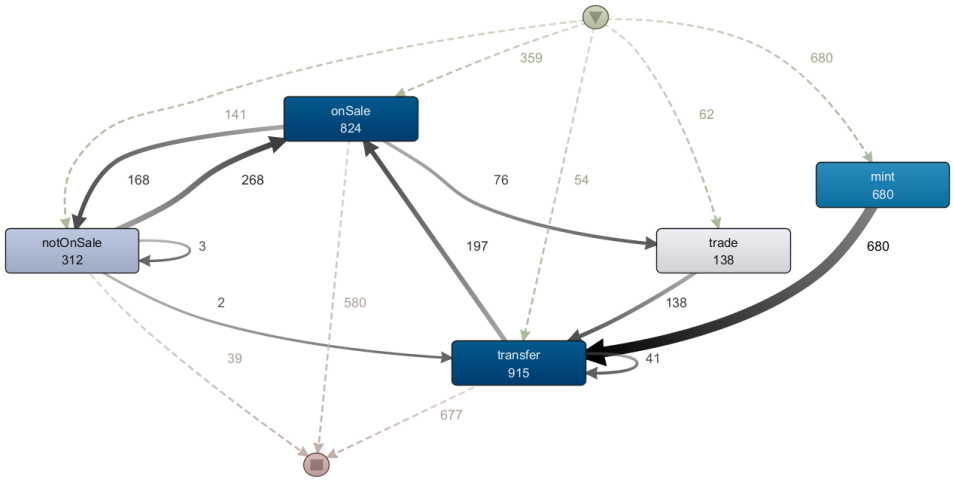
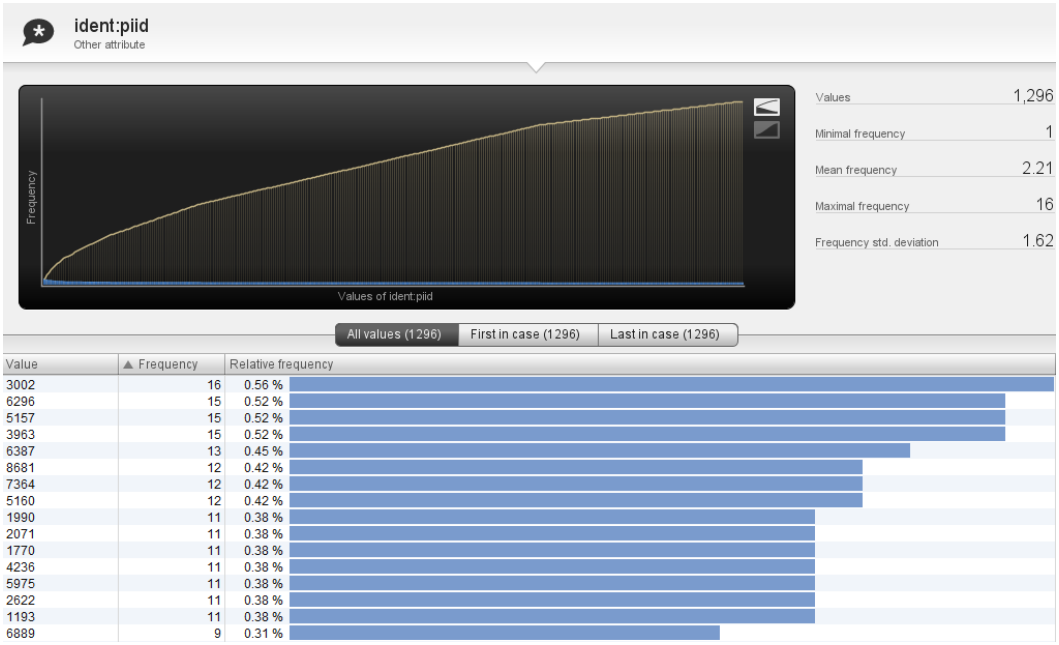


Figure 4.25. ETCBayc graph

The graph 4.25 was created from the XES file and shows the process model. We can see that there were 680 mints and for each mint 680 transfers, this indicates that when a new NFT is generated, the mint function calls the transfer function to transfer the newly generated NFT to the user. In just 22 hours 138 NFTs were purchased, once again for each NFT purchased the transfer function was executed. Of the 824 NFTs offered for sale, 76 were sold in just under a day. It can also be seen that the function TokenMarketPriceChange was not called during these 22 hours.

[25]https://github.com/IReallyLikeYourPants/Blockchain-Logging-Framework/blob/master/CASI%20PER%20BLF/4%20-%20ETCBAYC%20ETC/output4.xes



Value	Frequency	Relative frequency
3002	16	0.56 %
6296	15	0.52 %
5157	15	0.52 %
3963	15	0.52 %
6387	13	0.45 %
8681	12	0.42 %
7364	12	0.42 %
5160	12	0.42 %
1990	11	0.38 %
2071	11	0.38 %
1770	11	0.38 %
4236	11	0.38 %
5975	11	0.38 %
2622	11	0.38 %
1193	11	0.38 %
6889	9	0.31 %

Figure 4.26. ETCBayc statistics

Image 4.26 shows that a total of 1296 NFTs made at least one call to the smart contract, an average of 2.21 per NFT. The NFT that made the most calls was the *3002* one.

Figure 4.27. Representation of NFT 3002

Chapter 5

Conclusions and Future works

5.1 Conclusions

Thanks to this internship, we have built a tool capable of extracting event logs, CSV and/or XES, from Algorand and Ethereum Classic blockchains. The extraction was implemented on an already existing application called *Blockchain Logging Framework*.

The application provided enormous support regarding the implementation of ETC. To Algorand, it suggested an idea of the pattern to follow regarding the bcql manifest, which is necessary when choosing the data to fetch. After the implementation we tested our work to verify the correct functioning.

The analysed use cases are: the first, a simple one, useful to verify the transfer of ALGO; the second, to analyse the transfer of assets on the Algorand blockchain; the third, to investigate the extraction of data from smart contracts, also on Algorand; the fourth, to validate the correct implementation of the Ethereum Classic blockchain through the analysis of an NFT project.

The study of these use cases has also made us realise the limitations and possible improvements that can be implemented to the application. They will be discussed in the next section.

5.2 Future works

Blockchain Logging Framework is an indicatively recent application and in many aspects can be optimised. With regard to BLF and the changes I have made, the following issues can be implemented and resolved: add support for additional blockchains; implement a graphical interface to simplify the use of the application; facilitate the creation of the bcql manifest, which is currently a bit challenging; improve the data extraction mechanism, mainly regarding Ethereum and Ethereum Classic; add the possibility to generate as output XES files in relation to Algorand; improve Algorand's manifest syntax; implement the same syntax checking mechanism as for Ethereum for the Algorand manifest; add additional filters support for searching transactions through the Algorand Indexer.

Bibliography

[1] Paul Beck, Hendrik Bockrath, Tom Knoche, Mykola Digtiar, Tobias Petrich, Daniil Romanchenko, Richard Hobeck, Luise Pufahl, Christopher Klinkmüller, and Ingo Weber. Blf: A blockchain logging framework for mining blockchain data. 2021.

[2] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, pages 1–4. Chicago, IL, 2016.

[3] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[4] Claudio Di Ciccio. Towards a process-oriented analysis of blockchain data. In *Modellierung (Companion)*, pages 42–44, 2020.

[5] IEEE Task Force. Manifesto del process mining.

[6] Algorand Foundation. *Indexer*. Available at `https://developer.algorand.org/docs/get-details/indexer/`.

[7] Christian W Günther and Anne Rozinat. Disco: Discover your processes. *BPM (Demos)*, 940(1):40–44, 2012.

[8] Richard Hobeck, Christopher Klinkmüller, HMN Bandara, Ingo Weber, and Wil MP van der Aalst. Process mining on blockchain data: a case study of augur. In *International conference on business process management*, pages 306–323. Springer, 2021.

[9] Christopher Klinkmüller, Alexander Ponomarev, An Binh Tran, Ingo Weber, and Wil Aalst. *Mining Blockchain Processes: Extracting Process Mining Data from Blockchain Applications*, pages 71–86. 08 2019.

[10] Merit Kolvart, Margus Poola, and Addi Rull. Smart contracts. In *The Future of Law and etechnologies*, pages 133–147. Springer, 2016.

[11] Silverio Manganaro. Estrazione di event log per process mining da ledger distribuiti della piattaforma algorand. Sapienza Università Di Roma, 2021.

[12] Roman Mühlberger, Stefan Bachhofner, Claudio Di Ciccio, Luciano García-Bañuelos, and Orlenys López-Pintado. Extracting event logs for process mining

from data stored on the blockchain. In Remco M. Dijkman, Chiara Di Francescomarino, and Uwe Zdun, editors, *BPM Workshops*, volume 362 of *Lecture Notes in Business Information Processing*, pages 690–703. Springer, 2019.

[13] S. Bitcoin Nakamoto. *A peer-to-peer electronic cash system (2008).* Available at `https://bitcoin.org/bitcoin.pdf`.

[14] Marco Raffaele. Studio e realizzazione di smart contract su piattaforma algorand. 2021.

[15] Emanuele Rampioni. Blockchain e smart contract. 2020.

[16] Giorgio Rognetta. Crittografia asimmetrica: dal cifrario di giulio cesare alla firma digitale. *Informatica e diritto*, 8(1):59–75, 1999.

[17] Wil Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012.

[18] G Wood. *Ethereum: A secure decentralised generalised transaction ledger (2014).* Available at `https://ethereum.github.io/yellowpaper/paper.pdf`.