

BitSights



Analyzing the Bitcoin network

Nicolás Castano

Sebastian Kulesz

Abstract	3
Introduction	4
Related work	4
Technical Background	6
Architecture	8
Heuristics	8
Address Clusterization	8
Shared Inputs	9
Change address	10
Address Distance	11
Cluster Relationships	13
Wallet probability	15
Balance	16
Timed Balance	17
Volume	17
Engine Implementation	17
Internal representation	19
REST API	20
Data Visualization	21
Bitcoin Node	21
Node Hosting	23
Testing and Validation	24
Results	28
Demo - example of usage	32
Applications	39
Limitations and future work	39
Conclusions	40
References	42
Annex	43
Web application Implementation	43
Process	44
Materializing the idea	44
Designing process	45
Implementation	56
Skeleton structure	56

Main flow	58
Details pages	59
First implementation	59
Second implementation	59
Graph details	61
Graph visualization	63
Download graph	64
Heuristics	64

Abstract

Cryptocurrencies have fundamentally changed the way we do transactions. What once used to be a highly regulated interaction between businesses and/or individuals, has now become an uncontrollable market. Borders have virtually disappeared, and the requirement to share one's identity is now practically unenforceable by any institution deciding to participate.

This has created a lot of challenges in the form of fraud, thefts, untraceability, and several more. We will explain the anonymous nature of Bitcoin, backed by the Blockchain technology, and how it is guaranteed at a technical level. However, we will also show that there is side-channel information that we can exploit, in order to extract important information that can help us deanonymize a transaction, a wallet, or even a user.

Introduction

Blockchain technology brought us a revolution in terms of how money, or something of value for that matter, can be exchanged. Released in 2009 by an anonymous internet user going by Satoshi Nakamoto, the founding paper describes how it is feasible to build an online, distributed ledger where every transaction in history must be stored. Using the power of cryptography, it is ensured that transactions can be made peer-to-peer, without ever going by a centralized institution. This has created a lot of challenges for law enforcement agencies, where victims of fraud, or computer attacks, are left without any legal recourse that can be used in order to find the malicious party. A wallet is no more than a collection of addresses, each one identified by a random string. And this makes personal identification an impossible task.

With a permanent, immutable record of every transaction in history, where the possibility of being able to freely download it and verify its contents is a must, there is a way to analyze past behavior and extract important information that can help to build profiles, or even identify an individual or institution. Over the 10 years that the Bitcoin Blockchain has been online, over 200GB of information has been stored in it. Since January of 2009, more than 463 million transactions have been recorded, and several hundred thousand unique addresses have participated.

This makes available an incredible volume of data that is possible to analyze, transform, and finally extract all the generated metadata. Some heuristics that can be applied to the full blockchain, or to a particular address, will be presented in order to find related addresses and transactions that are not directly connected. This work will be integrated in a REST API that will be consumed by a frontend application which will display this information in the most significant form, and will allow a user to navigate its results.

Related work

While researching the state of the art for blockchain analysis, several papers¹² were found describing in one way or another, a similar set of heuristics. The base algorithm

¹ Evaluating User Privacy in Bitcoin: Elli Androulaki, Ghassan O. Karame , Marc Roeschlin, Tobias Scherer , and Srdjan Capkun

² Bitlodine: Extracting Intelligence from the Bitcoin Network: Michele Spagnuolo, Federico Maggi, and Stefano Zanero

implemented in this paper, which allowed the implementation of several others expanding on the information it provided, is an expansion in itself of the work described in some of those papers. This will be explained and expanded on in the following sections

Some services providing information similar to what is expected of this work were also found, although they were all paid and closed source.

Technical Background

Before diving into the implementation details of the heuristics previously mentioned, it is important to understand how the Blockchain is architected, and the technical background behind it.

What is important to remark, is that this database is distributed across thousands of contributing nodes, that do nothing more than storing the blockchain, and relying on it every time a new node joins. There are other fundamental operations that a node must do, which will be introduced at a later point. As was said, the Blockchain is a database. But, what kind of information does it store? And how is it structured?

As the name states, the Blockchain is nothing more than a series of **blocks**, cryptographically chained together. Each block contains a header which identifies it, with information like a checksum of its contents, the block number, and the amount of transactions present. Most importantly, it contains the checksum of the latest block present in the Blockchain at the moment that it was introduced into the database.

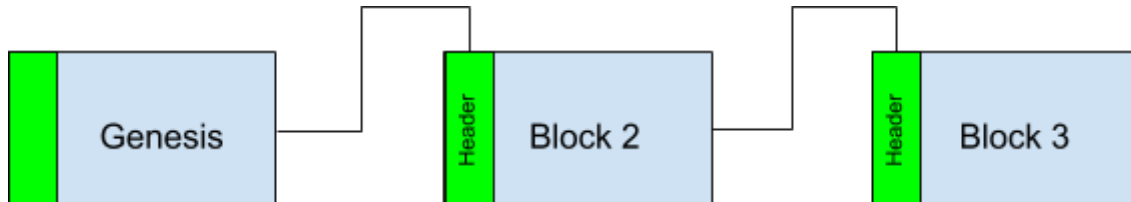


Image 1: Blockchain representation

If you haven't noticed by now, any change in a previous block will make its header change, and this will cause a cascade effect.

For example, let's say that a malicious actor tried to change block 2. Due to the nature of cryptographic hash functions, any change in its input can cause a completely different hash. So, if a change is applied, even so small as flipping a bit, the hash of block number 2 will be different than it previously was. As this value is then transferred into the header of block number 3, any subsequent block will suffer from a change.

But, detecting this tampering is what prevents the history to be rewritten. Once a block is added to the database, it is impossible to change it. And if someone tries to do so, it can be detected, stopping this malicious activity.

The most fundamental object that can be found in the Blockchain is an address. The most common form of it is composed by a pair of public-private keys. The public key is the string used to identify this address in the chain, for receiving BTC for example. While the private ECDSA key, generated from a 256 bits long random number, is used to cryptographically sign a message stating that a transfer is performed between a set of addresses, into another set of addresses.

The second most common type of address is known as a P2SH, where the “key” to transfer the funds is not the hash of a key, but rather the hash of a script. This allows for more complex transactions where one must know several key, a password, or anything to satisfy the execution of a script

Several times the transaction object was mentioned, so let's proceed to formally introduce it. As the building block of most cryptocurrencies, a transaction is a verified transfer of funds from a number of input addresses, to a set of output addresses.

Starting from the list of inputs, they are nothing more than previous outputs of past transactions, thus building a chain of signed transfers. Each input comes from an address, and contains a script that the node will use to test the validity of the transaction. Although extremely powerful, scripting is an entire area of study on its own, and is out of the scope of this paper.

Outputs are a similar structure, where one must list the recipient addresses and the amount that will be deposited on each. After being executed, a transaction will drain all the funds available from the previous output, so when transferring a fraction of the full amount available, the change should be sent to another address in control.

All this information is stripped of anything that can help us identify a subject. Neither addresses nor transactions contain personally identifiable information. And because of the algorithm used to build addresses, there is no useful information to pool the ones belonging to a single individual. The objective of Bitsights is to use the little data available, and do its best to enrich this information using the proposed heuristics.

Architecture

The application is architected as a decoupled web app, where all the heuristics are implemented and exposed via a REST API, and then visualized in a frontend in a way that can be understood and be of use for an user.

To query the Bitcoin Blockchain, a server was brought up in a public cloud. Because the size and information of the main network is so large, it becomes prohibitively expensive to download and index all the blocks mined. For this reason, it was decided to use the testnet for all our tests and demonstrations, although the mainnet could still be used if more resources were available.

This presented several advantages and opportunities for testing the proposed heuristics. Because of the existence of “faucets”, which are services that expend free Bitcoins for testing and development, it was possible to build a network of addresses and transactions exactly how the proposed heuristics would expect it to be for the information to be successfully extracted.

Using this fixture of data, it was validated that the algorithms were working correctly, and detecting the relationships as expected.

Heuristics

For the purpose of enriching the information that is already available in the blockchain, and to permit the user to find previously unknown relationships between addresses, several heuristics were proposed. These will be explained and will be visually walked through what the algorithm is trying to find.

Address Clusterization

With this set of heuristics the purpose is to group a list of addresses, possibly belonging to the same individual. As anonymity is a given in the Blockchain, it is almost impossible to infer who owns an address, or to whom a transaction is directed. The idea is to use the information stored in the transaction objects to build a set of addresses in which the private key used to sign the transactions must be in the hands of the same individual.

Shared Inputs

The most basic heuristic is the one in which the link between two transactions is the presence of the same input address. The reason for this is that if one input is present in more than one transaction, it is asserted that the individual controls the private key of the rest of the inputs present.

To understand this, let's first see the following chains:

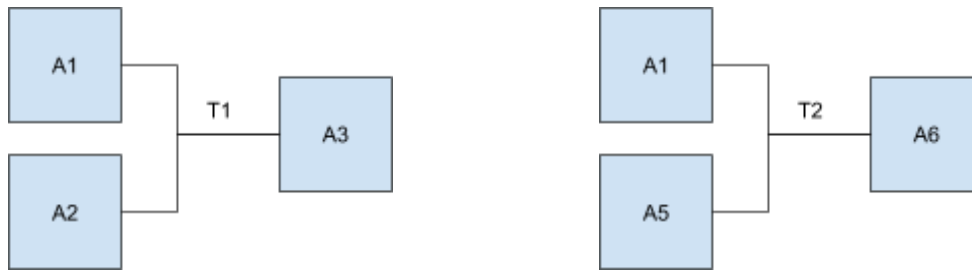


Image 2: Shared inputs

Focusing on the first transaction, where two outputs are transferred into a single address, it is inferred that the private key of both addresses from which the funds are being taken from must be in the hands of the same individual.

Let's say that addresses A_1 and A_2 are transferring funds into address A_3 via the transaction T_1 . As both addresses A_1 and A_2 are inputs in this transaction, the private key of both are in the hands of the individual I.

It is possible now to query the blockchain for all the transactions in which address A_1 was involved, with the exception of the one recently analyzed. Let's say it was found that it was also an input in transaction T_2 . This must mean that this individual must also be in control of the private key of address A_5 .

This process could be repeated until it is not possible to add any other address to this set of related addresses.

In the end, it is known from two previously unrelated transactions that the set of addresses $\{A_1, A_2, A_5\}$ must be controlled by the same individual.

Change address

This heuristic has its bases in how most of the modern wallets work. When the user wants to perform a transaction in which only part of the balance available in an address has to be used, the remaining must be transferred into a new address also in the control of the user for further use.

Let's try to further explain this with an example. Assume individual I_1 wants to transfer 1 BTC to individual I_2 . To perform this transfer, I_1 will use the 2 BTC it has available in address A_3 . Also, it was previously mentioned that when an address is present in a transaction as an input, referencing a previous output, the full balance of this output must be spent. To represent this visually in the image below, the *Transaction 1* corresponds to how the individual I_1 got the funds, and *Transaction 2* to how they are transferred to I_2 from the perspective of I_1 .

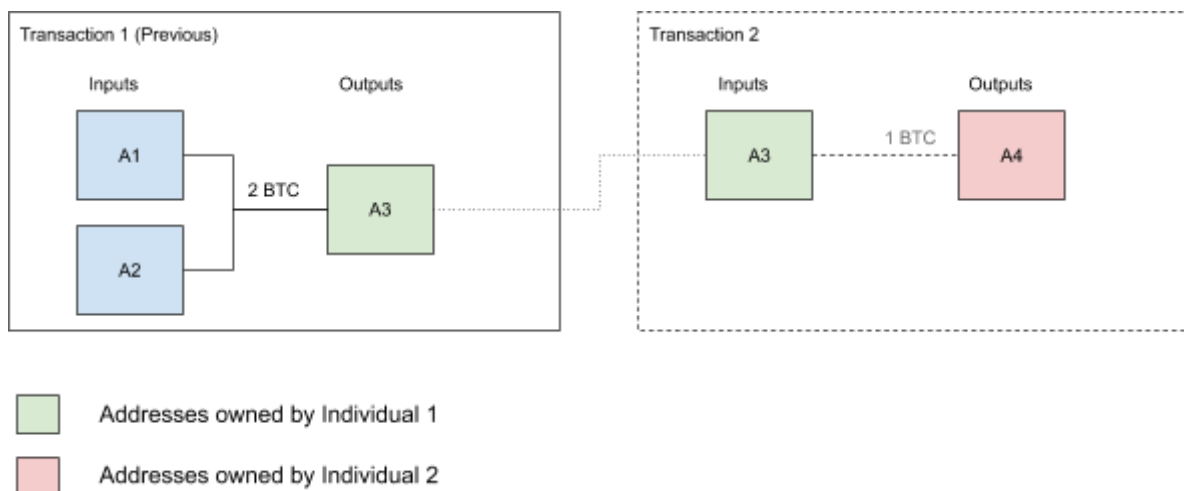


Image 3: Partial funds transaction

This creates the problem that if the individual only wants to transfer part of the funds, it is needed to somehow receive the rest so that it is feasible to spend it in the future. This is what the wallets refer to as the change address, and is what is expected to be detected in order to add this address to the related set, as the private key of the change address will also be under the control of the same individual that created it.

Following the above example, represented in the image below, imagine an individual wanting to transfer 1 BTC from the 2 BTC that address A_3 has available. To achieve this, the wallet needs to transfer the remaining 1 BTC (assuming that there are no fees) to another address so that it is attainable to use it in the future. Therefore, the real chain would look like the following image.

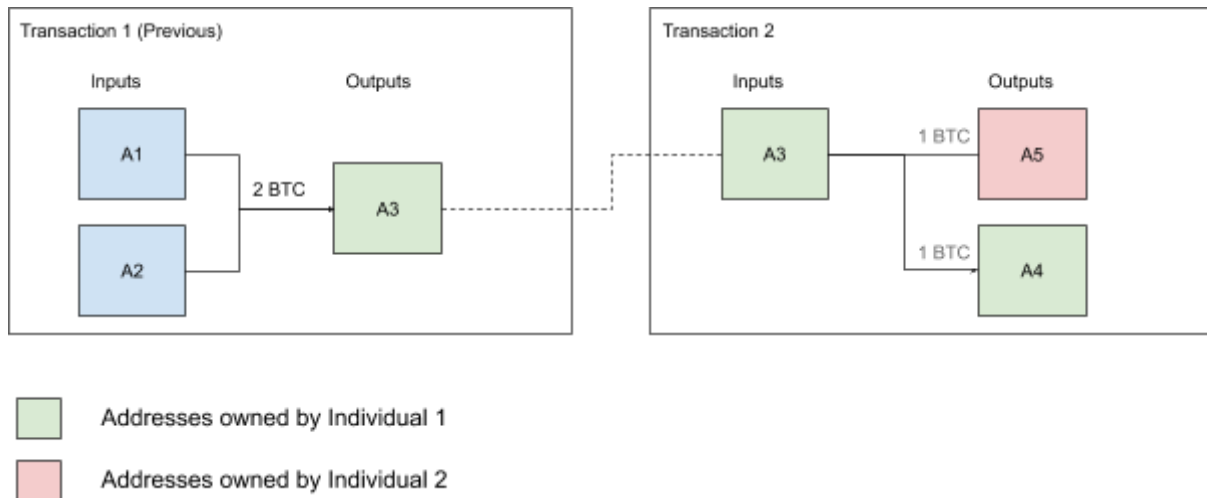


Image 4: Change address

The goal then, is to find this address A_6 and add it to our set of related addresses for individual I_1 . It can be stated that the private key will be in their hands as the wallet has created this address.

Address Distance

Another piece of information which may be of use, is how further apart are two addresses in the blockchain, considering as distance the amount of transactions in between. Although innocuous at first, this information may be of use to relate two previously unrelated clusters of addresses. There are some websites which offer the service of scrambling the source of a certain amount of funds by generating bogus transactions. It is possible this heuristic, along with some other external source of information, to see how far away these addresses are.

The functionality is really simple, as it is only needed to provide a source address to start the search algorithm, and a sink address to stop once it is found. A third parameter is the depth limit, which is left to the user to decide.

The system will then perform a BFS search, going over the transactions in which each of the addresses was involved, prioritized by the distance to the source address.

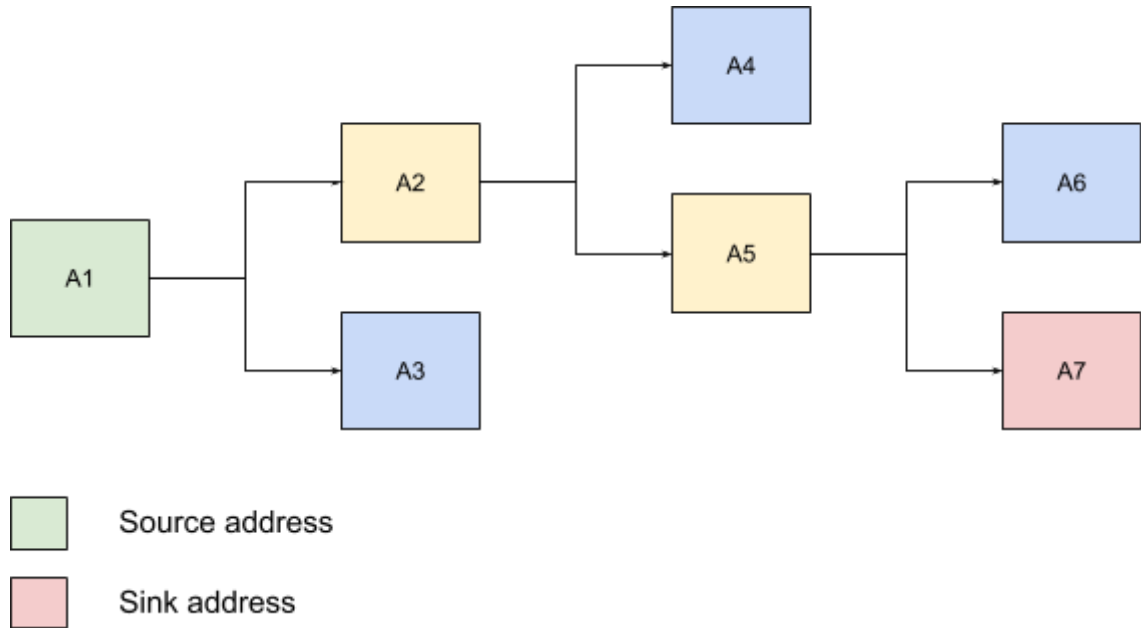


Image 5: Address distance

In this example, the objective is to study how many addresses are needed to traverse to get from address A_1 to address A_7 . Each edge represents a transaction between an input, and two outputs. It is important to mention the fact that this simplification will be utilized in order to represent an ideal situation.

The algorithm will start by finding all the transactions in which A_1 participated, and iterate over them finding the sink address, or adding them back to the queue of transactions required to be processed. This queuing is what will cause the path in which the addresses are traversed to be equivalent to a DFS search. This can be evidenced in the numbering of the address of the example, which coincides with the order in which they will be analyzed.

Having a more complicated case, where a loop exists in the “tree” (now a graph) which is being traversed, it will end returning only the shortest path that exists between the source and the sink.

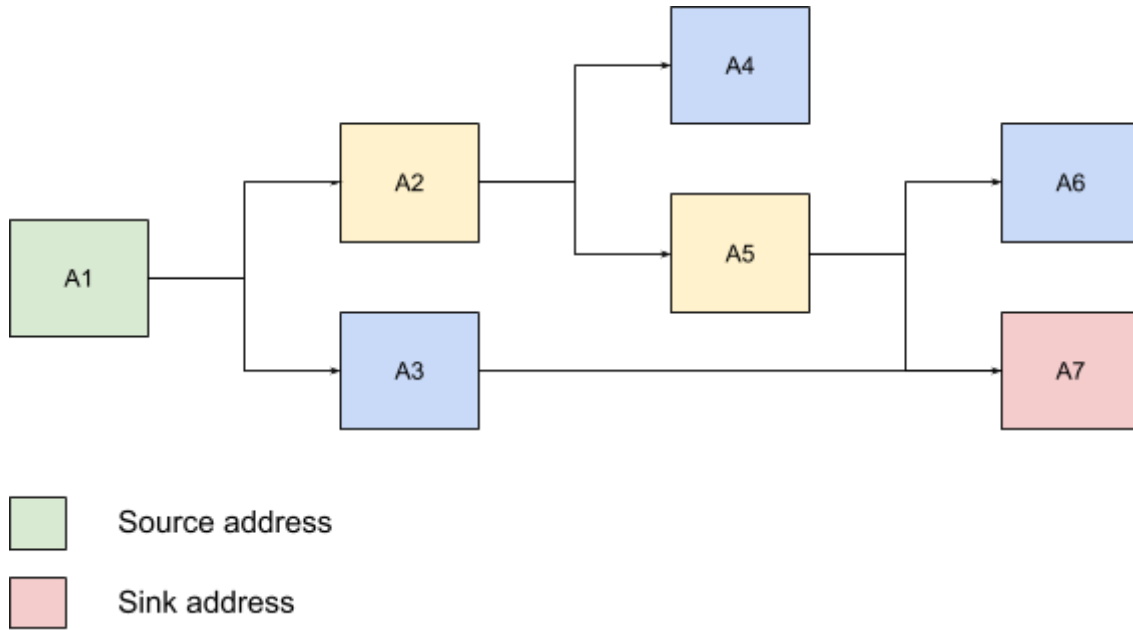


Image 6: Address distance with a loop

In this example, let's assume that there exists 2 transactions, T_1 and T_2 , in which T_1 transfers funds between $A_3 \rightarrow A_7$, and T_2 between $A_5 \rightarrow A_7$. This case is common for public addresses used for receiving donations for the public.

As tree is being traversed by levels, the algorithm would first see the transaction T_1 , and never realise that there was another (longer) path if it had followed another branch. Because the heuristic returns the shortest paths between two addresses, this was decided to be a non-issue.

A key problem to mention, is a transaction in which several tens of addresses are used as outputs. In this situation, as a lot of branches are created, the heuristic is not smart enough to discard paths that will end up nowhere.

Cluster Relationships

This heuristic was born as a derivative of the "Related Address" heuristic. It is meant to find how related two clusters of addresses are, and returns the edges that cross this boundary. As it is best seen with an example, let's build one. As an input for this heuristic, two addresses are necessary, A_{11} and A_{21} , which will serve to build both clusters of related addresses.

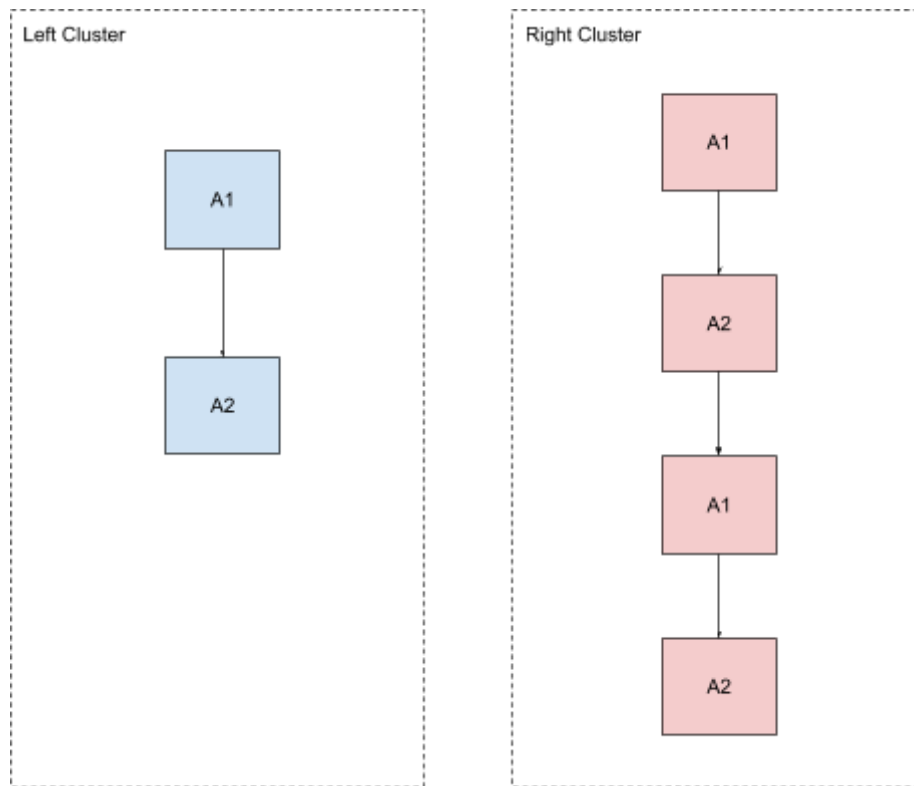


Image 7: Cluster relationship

It is important to note that the direction of the edges does not necessarily denote a transfer of funds, but the path in which the heuristic discovered the association between both ends of an edge.

Once both clusters are built, and all other addresses related to the initial ones are found, it is essential finding in which transactions all these addresses participated, and see if any of this has an input belonging to one cluster, and an output belonging to the other.

To achieve this, a list was built with all the transactions that are present in the left cluster, and another list was created for all the addresses that are present in the right cluster, and then run the following algorithm twice, once for each list.

The first time it runs, the algorithm will be iterating the first list of transactions, looking for one that crosses the boundary from left to right, that is, that has an input in the right cluster and an output on the left cluster. This is saved as an edge that goes from left to right. Then, the same algorithm is executed, but inverting the order of the parameters to find the cross-transactions that go from right to left, and save the result in the same format.

The final result, when graphed, has the following appearance:

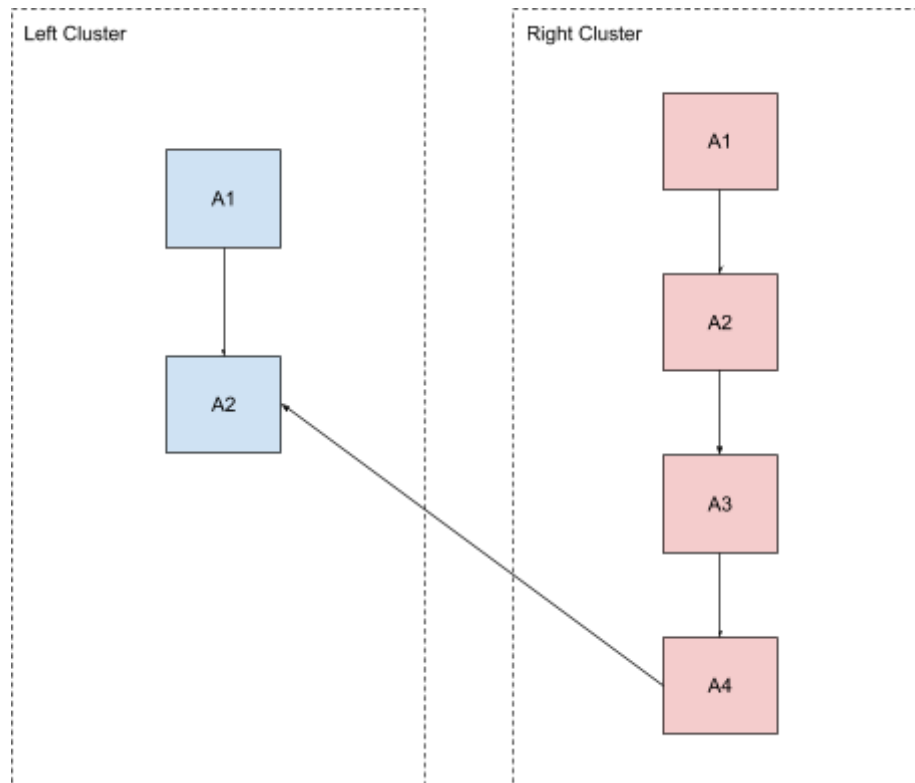


Image 8: Cluster relationship with cross-edges

Wallet probability

One big source of information on the type of individual that owns a cluster is the software used to manage it. Although the specific application or version can not be pinpointed, it is viable to calculate the probability of it being a Wallet, or an automatic algorithm probably used in an exchange.

It is possible to calculate this probability by looking at the number of outputs of the transactions that have any address of the cluster as an input. It is expected that any transaction made by a human will probably be a 2-party transaction. Funds will be transferred from an address, or set of addresses if the balance of one is not enough, to a unique address. Probably to pay for a product or a service. It is also known that automated software will try to minimize the fee as much as possible, so an exchange may choose to mix several inputs with several outputs in the same transaction.

This gives an easy metric to calculate. Over the population of transactions in which the addresses of a cluster was involved, if the total of transactions with 1 or 2 outputs (to account for the change address) divided by the total amount of transactions is over a certain value, it can be said that, with a certain degree of probability, the cluster is managed by a

Wallet. If this value is below the threshold, it probably belongs to an automated application or exchange.

Balance

Calculating the final balance of a cluster can be a tricky operation. Imagine a set of addresses that are all managed by the same individual: how it is possible to detect what inputs and outputs to take into account?

More specifically, imagine a transaction with 2 inputs. One originated as an output from an address that also belongs to the cluster, and the other from an address external to the cluster. How can the system decide if they both should add to the balance, subtract from it, or leave it unchanged?

The following rules were proposed to calculate the balance:

- If an **input** originated from an address **not present** in the cluster, it **adds** to the balance
- If an **input** originated from an address **present** the cluster, the balance is **not modified**
- If an **output** is directed to an address **not present** in the cluster, it **subtracts** from the balance
- If an **output** is directed to an address **present** in the cluster, the balance is **not modified**

These must be applied to every transaction in which the addresses of a cluster participated. As a result, a very close estimate of the balance will be gotten, only affected by the difference paid in fees. Another set of rules could probably be proposed, but it was decided to be good enough to get a close estimate of the real balance the individual would be able to manage.

Although a trivial balance calculator would use the UTXO of the cluster of addresses, that is, the sum of outputs that have not yet been spent, using this method allows the construction of the following heuristic.

Timed Balance

Graphing the balance of a cluster over time could give potentially valuable insights. This heuristic will grab the set of rules mentioned above, and apply them to an ordered list of transactions, sorting them by the time in which the block that includes them was mined.

For each transaction, a partial balance will be stored, along with the timestamp. This will be returned so that a timeline graph can be built from this data.

This heuristic will calculate the balance at every point in time that a cluster “owner” has transacted. This is achieved by building a set of all the transactions in which the addresses of the cluster participated, and adding all the incoming and outgoing volume. The value being left with is the total volume of Bitcoins that flowed into the cluster, along with the volume that flowed to the outside of the cluster.

Note that the difference of these values most probably will not be the final balance of the cluster, as only inputs and outputs that belong to addresses from inside the cluster are considered.

An example on how this information is displayed to the user will be shown at a later section.

Volume

This heuristic, similar to the one just described above, will calculate the net volume of Bitcoin that a cluster has received, or sent to an external address, at every point in time where a transaction from said cluster exists.

Engine Implementation

The implementation of the engine proved to be a challenging task. After an initial implementation in Python using the Command pattern, it was completely rewritten from scratch in TypeScript to take advantage of the Promise system that Node.js has to offer. This allowed the team to scale better, and iterate faster over the implemented heuristics, as well as to produce much clearer code.

With the insights gained with the first implementation, the decision to rewrite the engine in TypeScript was taken. Two big advantages lead us to this: the strong type system

that the language has to offer, and the Promise pattern which is heavily designed around IO blocking operations.

This Promise system allowed programming the algorithms and heuristics as if every call were made synchronously, when, under the hood, every operation was handled asynchronously. This synchronicity when returning from a function is possible by returning a promise, instead of a real value. This promise will eventually be resolved to what the caller is actually expecting. Following this pattern, the system can ask an heuristic, for example, the cluster of related addresses given a single one. The system will return a promise which, when resolved, will contain the list of addresses that conform the cluster. And will only resume execution from when the call was made, when the result is available, and not before..

Internally, once an initial address is received, it will then use the heuristics at hand to obtain the first level of addresses related to the initial one. For each of this addresses, the same query is performed to the system as if every one of these were the initial address given to us.

Following this process recursively, it is seen that there is an implicit tree of promises. Where each level is built with each node of the previous level as if it were a needle address.

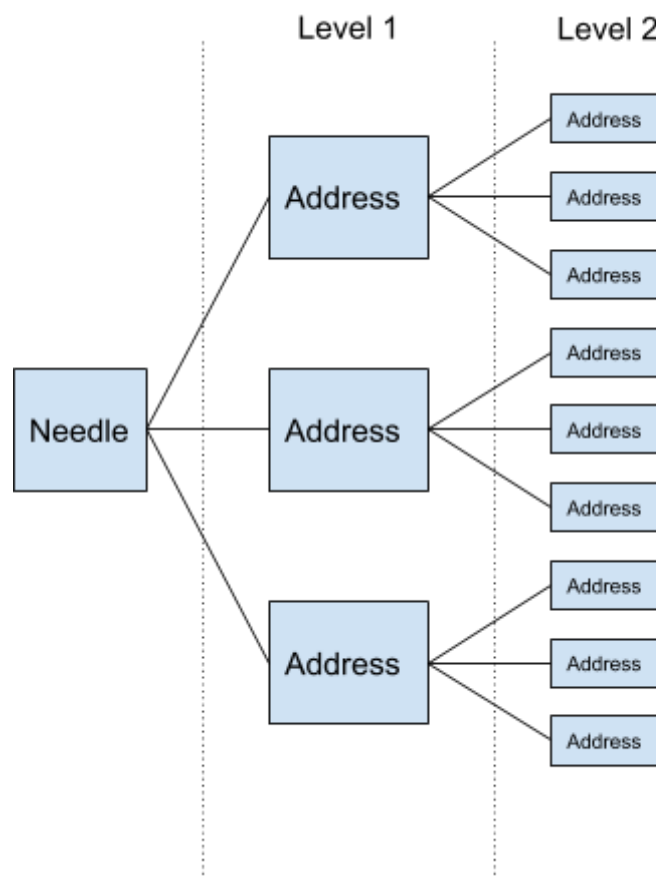


Image 9: Implicit promise tree

Following this example, and imagining that the level 2 is the last reachable layer (that is, no more transactions relating additional addresses can be found), using the promise system enables us to just wait on the result of the first level, and let the system notify us when a result is available.

Let's follow the tree flattening algorithm to see how this process works. It starts by submitting a needle address to the system, and in return, a promise of a list of addresses is gotten. As this is the needle submitted by the user, this answer of this promise will be the one returned back.

Once this needle address is submitted, the following algorithm is executed:

1. Get all transactions in which this address participated.
2. For each of these transactions, start from step 1 and create a promise for the result that will eventually be available. (Notice how this algorithm is executed concurrently, and only promises of future results are being manipulated)
3. Wait for all the promises created at this level, grab the results of each one of them in an array, and flatten it.
4. Return the results array as the value for the promise that is waiting one level lower than us.

Notice that on step 2, where the recursion takes place, if no more addresses are found, this means that the current node is a leaf, and thus an empty array is returned. If it were an intermediate node, step 3 would wait until the upper levels are resolved.

Under the hood, this implements an DFS search algorithm, where the order in which the nodes are visited is given by the promise resolution system.

Architecturally, the system is composed by a set of providers, abstracted by a repository. This gives the user the possibility of choosing at launch time the source of the information that will be queried. Noted, for any new source, the provider will have to implement the required queries, and transform the information into the models used internally.

Internal representation

While a Bitcoin transaction exposes a lot of information, only a subset of it is required to properly apply the heuristics. For the purpose of reducing the memory usage as much as

possible, and limiting the amount of information being shuffled in the heap, most of this information will be stripped by the providers making the actual request to the server.

For the current implementation of the system, given a complete transaction, only the inputs, outputs, hash, and mining time are required.

```
export class Transaction {
  public readonly inputs: TransactionAddress[];
  public readonly outputs: TransactionAddress[];

  public readonly hash: string;
  public readonly time: number;
}

export class TransactionAddress {
  public readonly address: string;
  public readonly value?: number;
}
```

This is enough information to build the tree, find the relationships, and return the queried information.

If future implementations or heuristics happen to require additional fields, the system can be easily adapted. Special care must be taken with this procedure as not every provider could be exposing this information. As a premise, every field used by the heuristics must be available in every provider, and properly adapted from the network response, into the provider's response.

REST API

To interface with the engine, a REST API was built. With the goal of it being as simple as possible, only 3 endpoints were created: One to create a job, one to query the status of a job, and one to extract the job results.

To create a job, it is first necessary to select the heuristic that should be applied, as every one of them accepts a different set of parameters. Once decided, a POST with a JSON specifying the heuristic, along with the parameters should be sent.

```
{
  "args": {
```

```
    "needle_address": "n45tjXVF3KdTcgWtHo41vXCyXXZWL8D9bg"
  },
  "job_type": "RELATED"
}
```

As a response, a status code of 202 CREATED along with a Job ID will be received. This ID can be later used to query the state of the Job:

```
{
  "status": "running",
  "type": "RELATED",
  "uuid": "a07b05fd-882b-4b71-9c8c-fba7d4967333"
}
```

Once the status field value changes to *finish*, the results endpoint will be available to fetch the result in two possible formats. While a JSON document is the default, for some heuristics it is possible to transform the result into a Graphviz DOT document, so that it can be directly graphed using the proper tool.

Data Visualization

For the purposes of allowing the visualization of the information generated by this system, a web application presenting the graph in a significant way was developed. The user is able to input an address, or the parameters that the specific heuristic is expecting, and once the result is available, be shown the information with the most appropriate graphing solution. More information about the development of this application is included in the Annex.

Bitcoin Node

Having implemented all the heuristics and APIs needed to perform the analysis, it is now needed to somehow collect information in a structured way, giving the system the possibility to query it. As of the moment of writing this paper, the Bitcoin Blockchain alone weighs more than 270 GB. And although cumbersome, it's not unmanageable. Also, there are several open source clients which allow us to download it, and even actively contribute to its network.

However, a simple copy of the Blockchain would not allow us to do the required queries. The way most of the clients download the blocks and assemble the index, do not allow queries in the form of “Give me all the transactions in which this address was a participant”. Instead, they just allow access to a given transaction with a known hash. This functionality is needed so that the heuristics can explode a node into several others, by knowing every transaction in which an address participated. A project with a fork of the official Bitcoin client was discovered, but this was long abandoned, and lacked support for the latest features of the protocol, meaning that queries for newer blocks could return incorrect results.

An alternative for an in-house solution would be using a relational database like PostgreSQL and a simple schema which would make this solution simple enough to implement. But, with the volume of data that needs to be managed, the index generated by this tool would be prohibitively large to manage. And outside of the resources available to the team.

After discarding this option, several online Blockchain explorers were found with public APIs, most of which implementing the required search functionality. However, as the tree being traversed grows fast, rate limiting prevents the system from fully utilizing only one provider. To bypass this limitation, a set of providers and adapters was designed so that all of the services providing the functionality that was required could be queried at the same time.

The idea was to transform all the different responses from these services into a common scheme, to later be consumed by the heuristics. Some limitations to this system that were found related mostly to rate limiting still being applied to the queries the system performed, and the impossibility to do paged lookups for when the responses were large enough to not fit in only one response.

The doubt, then, was on what systems do these Block Explorers base their search engines. After much research, a solution called BCoin was found. This open source project implemented a full Bitcoin node in Javascript, with the added possibility of generating a full address to transaction index. This index, however, would still be prohibitively large to manage and store on the cloud, so the decision to index the testnet instead was made.

The testnet, as the name implies, is a parallel chain used for testing and developing purposes. It has the same structure and follows the same rules and protocols as the mainnet, but the holdings of a wallet are worthless. The main benefit of this decision was

two-fold. First, the testnet is only a few gigabytes in size, so the fully indexed chain would not weight that much. Second, and most importantly for the reasons of this project, it allowed the team to use what are called Bitcoin Faucets to test the heuristics. Faucets are projects which distribute free bitcoins on the testnet so that developers can test and validate their projects. Using these systems, the team could test and validate the heuristics on hand-made transaction trees, made specifically to account for all the edge cases which could be found.

During the investigation, an explorer named Blockchair was discovered, providing access to perform SQL like queries to the blockchain. After communicating with them, they were keen to provide an API key allowing access to the mainnet, and enabling testing using real, publicly known addresses.

Node Hosting

Hand in hand with the decision on the querying system detailed on the last section, the decision on where to host the data had to be made.

From the analysis of the different solutions, it was estimated that around 750 GB of information had to be downloaded in order for the Blockchain and required indexes be built. This resulted in a cost comparison to have, at least, and accurate estimation of how much it would cost to run this system indexing the mainnet, and how long it was possible to host this solution using the free tier offered by some cloud providers while indexing the testnet.

As a baseline, and extrapolating the data gathered using a subset of the mainnet, these are the resources required to run the solution with a basic SLO (Service Level Objective) ensuring appropriate response times, and a TTM (Time To Market, or time until the full blockchain can be queried) of 2 weeks.

Testnet:

- Storage: 40 GB
- CPU: 4 VCPU for indexing, 2 VCPU for querying
- RAM: 4GB
- Initial ingress: 27 GB

Mainnet:

- Storage: 2000 GB
- CPU: 4 VCPU for indexing, 2 VCPU for querying

- RAM: 16GB
- Initial ingress: 750 GB

As previously mentioned, the testnet is only a fraction of the size as the mainnet, so the resources mentioned are the minimum to maintain acceptable query times. With this specifications, it is possible to sync and index the full testnet in a matter of hours. In the server used for this project, the entire process took 6 hours.

Results for the mainnet are an extrapolation of an initial sync, only reaching a few percentage points. As the usage that this chain had in the early beginnings is not the same as the one it currently sustains, these values should be taken as a baseline.

These requirements were used as inputs for the price calculators that most cloud providers make available to the public. These were the results for the most popular providers:

- AWS: U\$D 325 / month
- Azure: U\$D 380 / month
- GCP: U\$D 437 / month

Testing and Validation

Given that properly testing and validating that the implemented heuristics are only returning the correct results is hard to accomplish by only executing unit or integration tests, advantage of the Bitcoin Faucets was taken to build a model chain of addresses and transactions.

Using several Faucets, and an advanced wallet which allows the user to manually select the inputs and outputs of each transactions, the following chain was built:

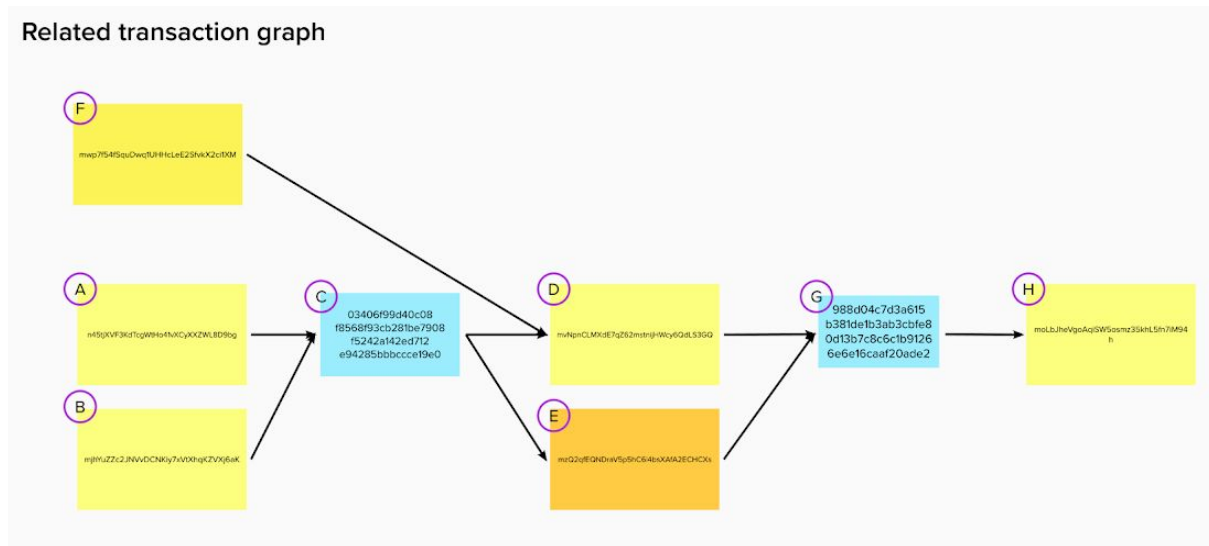


Image 10: sample graph

This graph, read from left to right, describes transactions made in time, specially crafted to test all cases that the heuristics must take into account to relate addresses that belong to a single entity.

In this graph, blocks with a shade of yellow correspond to addresses, and block with a shade of light blue correspond to transactions.

Block F is a transaction that originates from a Faucet, having as output the address D. Blocks A and B correspond to addresses previously funded using a faucet, too. Starting with the first transaction, block with label C, containing two inputs from block A and B, and having two outputs. The edge C-D corresponds to the output going to the address chosen by the team using the wallet software. Edge C-E, on the other hand, corresponds to the change address where the remaining funds were transferred, as the sum of the unspent outputs of addresses A and B exceeded the amount being transferred to D.

Next, funds from addresses corresponding to blocks D and E are transferred completely to block H through transaction G.

If the Address Clusterization algorithm were to be run on any of the addresses from block A, B, C or D, the following response would be received:



Image 11: graphviz visualization

For reasons of brevity, only the first four characters of the addresses and transaction will be used to reference them.

This execution of the algorithm was given the address *n45t* as a seed. The edges, in this case, do not mean an exchange of funds, but rather the direction in which the heuristic has found the relationship. This means that address *mjhY* was found to be related to address *n45t* through the transaction 0340. As can be seen on the graph above, both addresses are inputs to the same transaction, and it can be assumed that the private key is known by the same individual.

Also, address *mjhY* is related to address *mzQ2* through transaction 0340. However, one was an input to a transaction, and the other an output. What the heuristic found was that address *mzQ2* at this transaction, is the first time it is seen in the blockchain, and thus it is new. The conclusion is that this address was generated by the Wallet software as a change address to transfer the remaining funds, and the private key would also be in control of the same individual that initiated the transaction. To visually differentiate this relationship, an orange edge is used to graph it.

As was said, the same result should be returned if any of the four mentioned addresses is tried. The difference will reside in how the system graphs the relationship between them.

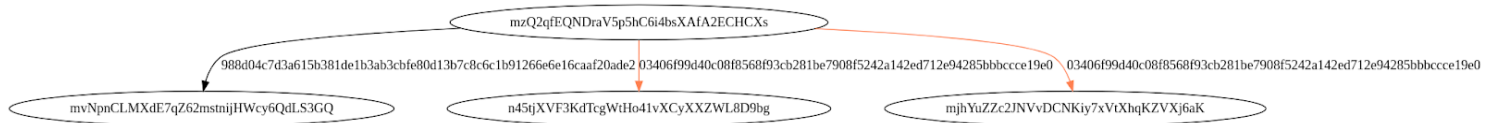


Image 12: graphviz visualization

In the image above, starting with address *mzQ2* , which is a change address, it can be seen how the system detects this, goes backwards into the transaction chain, and adds the addresses from the transaction that originated it.

Results

Using the network of transactions built for the testing and validation phase, several tests were performed. As could be seen in the previous sections, the clusterization algorithm successfully allows the user to find any address related to a needle by applying some heuristics to the transaction graph publicly available in the blockchain.

As also mentioned, using real data from the mainnet became an impossibility because of the size of it, so taking advantage of the Faucets, several models were built to analyze them.

Using the cluster relationship heuristic, a second, smaller cluster than the examples used for the previous sections was built. Then, a transaction to this cluster was performed, in order to establish a cross-cluster relationship. As a result of this transaction, the following edge can be obtained when checking for this kind of relationship.

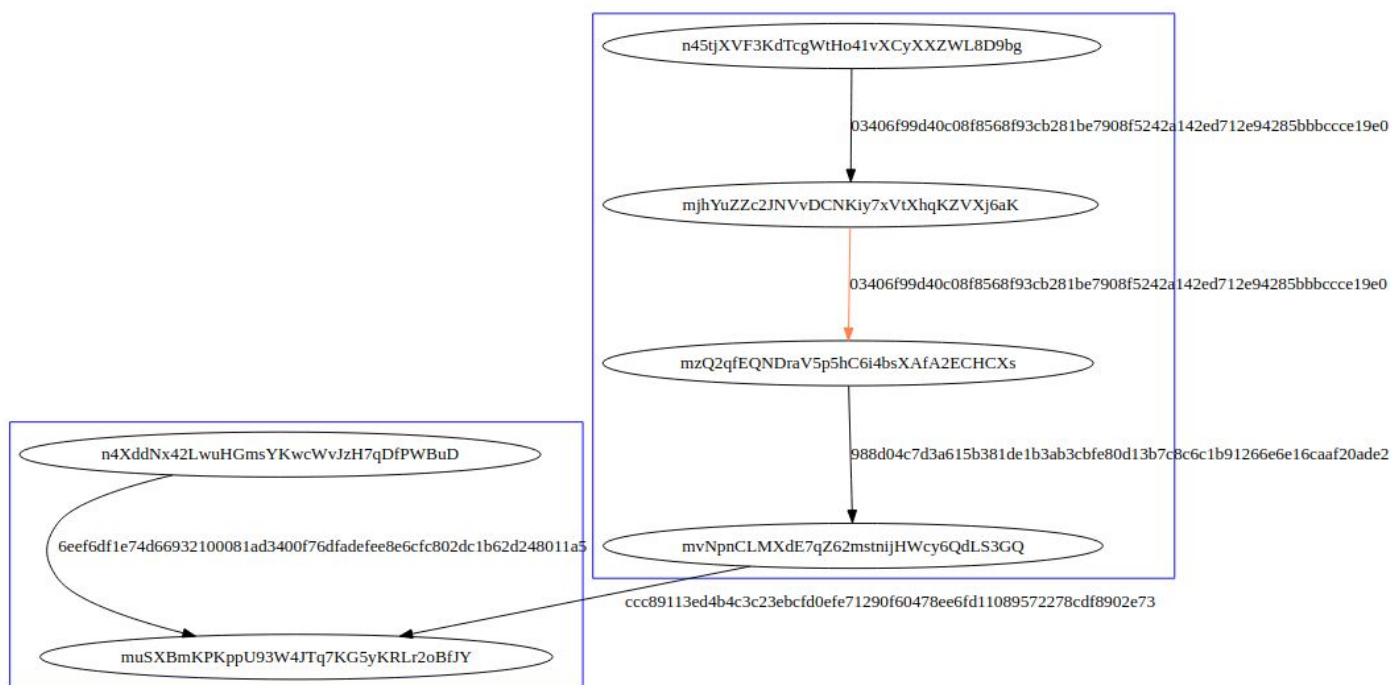


Image 13: cluster relationship example

Using all the transactions needed to build this graphs, it is also possible to calculate the balance as a function of time for the bigger cluster:

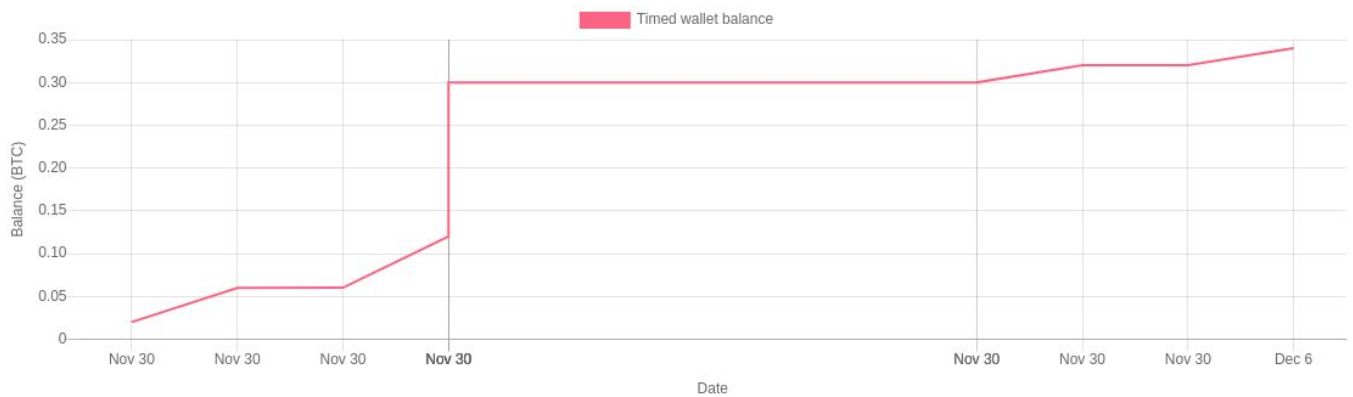


Image 14: wallet balance example

In this graph, it is possible to see how the balance of the cluster grows as more funds are transferred from the Faucets.

In order to test behavior of web application, several models were selected by collecting data obtained from a clusterization script, which performs an analysis of an arbitrary number of blocks (last 20 blocks by default) using bcoin client. As a result, the script prints some statistics like an histogram of cluster sizes, and the addresses that make each of them.

In the cases where graphs exceeded 100 nodes, it was harder to construe nodes connections due to the amount of edges involved, specially in Clusterization graphs. Despite the application offering the possibility of zoom in and out the graph, it was not easy to distinguish specific data. Besides, response time increases when having big amount of nodes and edges, not only because server needs to query bcoin client but also because the data response from server needs a preprocessing step before being displayed. Basically, this pre-processing then is reflected in *Details* section, when a node is selected it is then possible to see which nodes are connected to it.

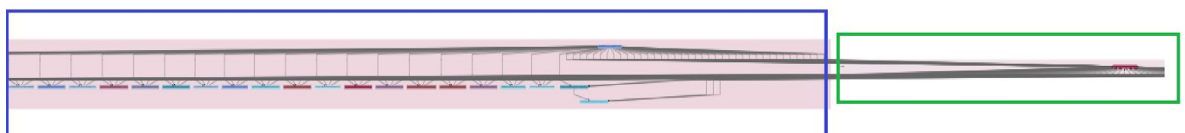


Image 15 - Zoom out of a two-cluster graph with 28 nodes and 127 edges.

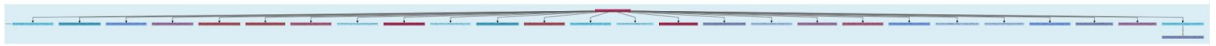


Image 16 - Related graph with 28 nodes and 28 edges.

Another important thing to take into account is the fact that for big graphs, it is not possible to download .png images because of limitations imposed by the used library.

On the other hand, the addition of a configuration sidebar in the graph section helps users to have a better experience when manipulating graphs.

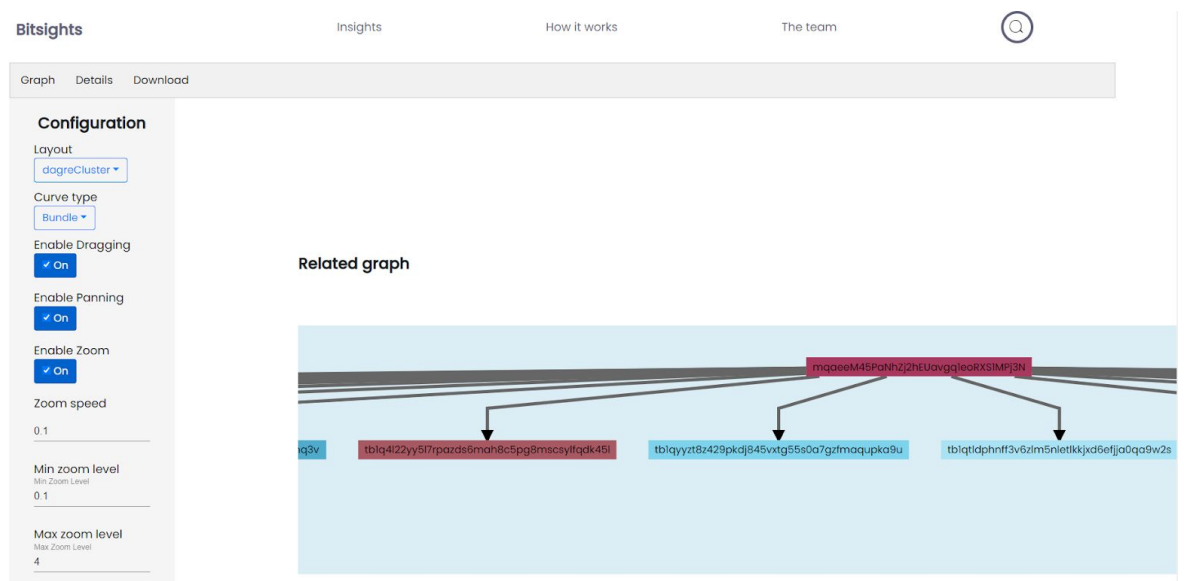


Image 17 - Example of Graph section tab, showing Configuration sidebar. It allows users to get a more friendly experience.

And for those cases where graphs are huge, *the Details* section helps to get further insights of transactions and addresses such as information obtained from block exchanges, as well as metrics previously mentioned.

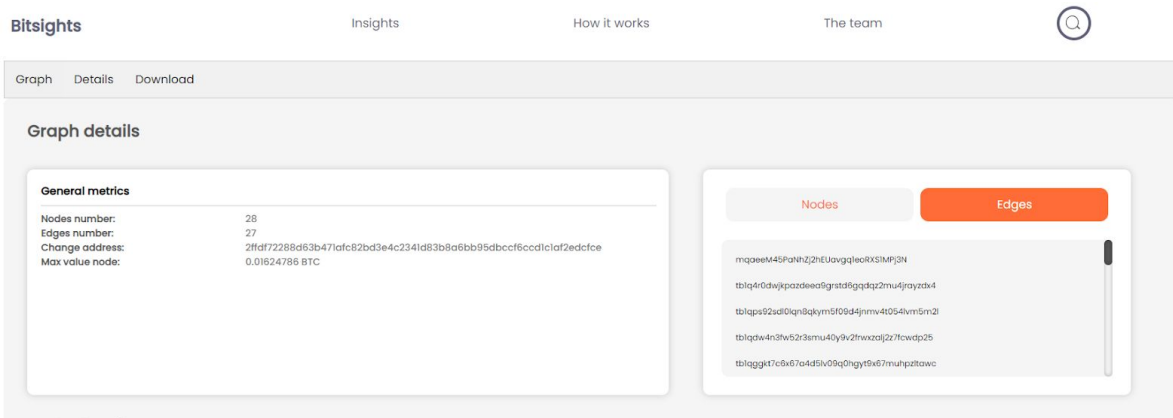


Image 18 - Snapshot of Details tab section.

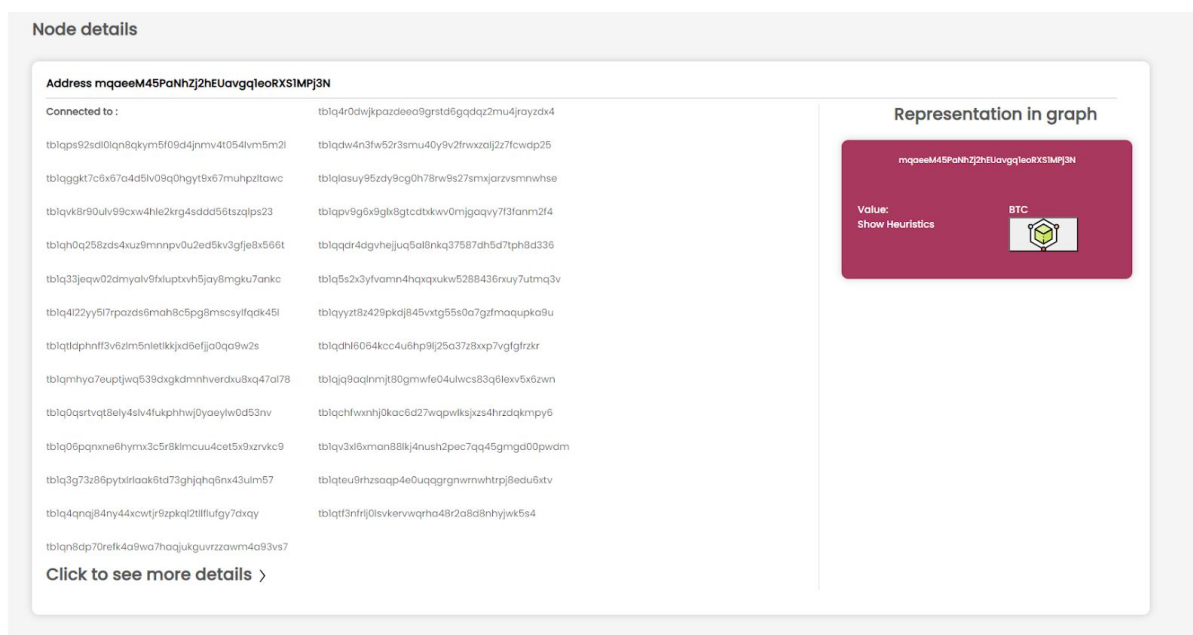


Image 19 - Snapshot of Node details section

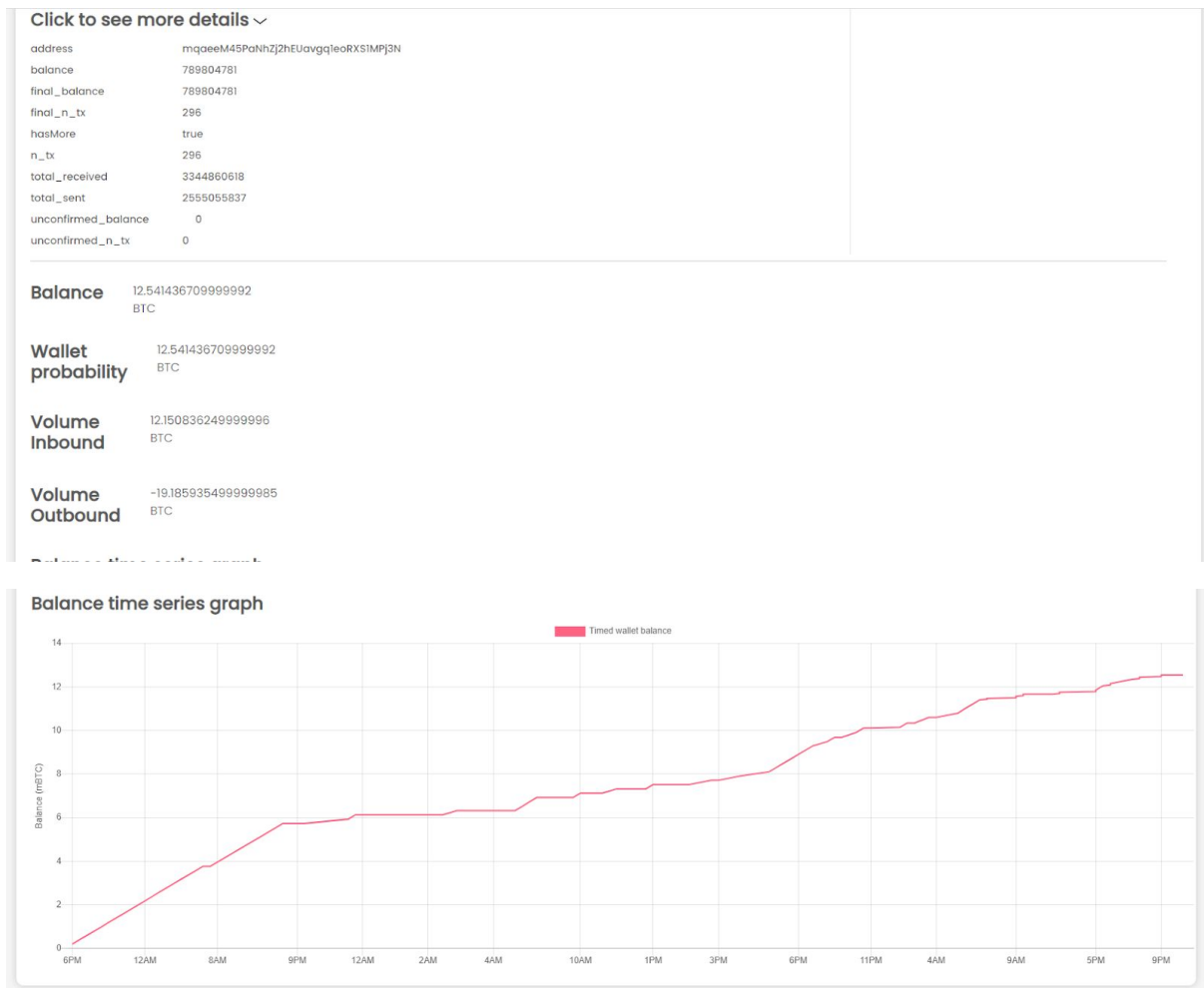


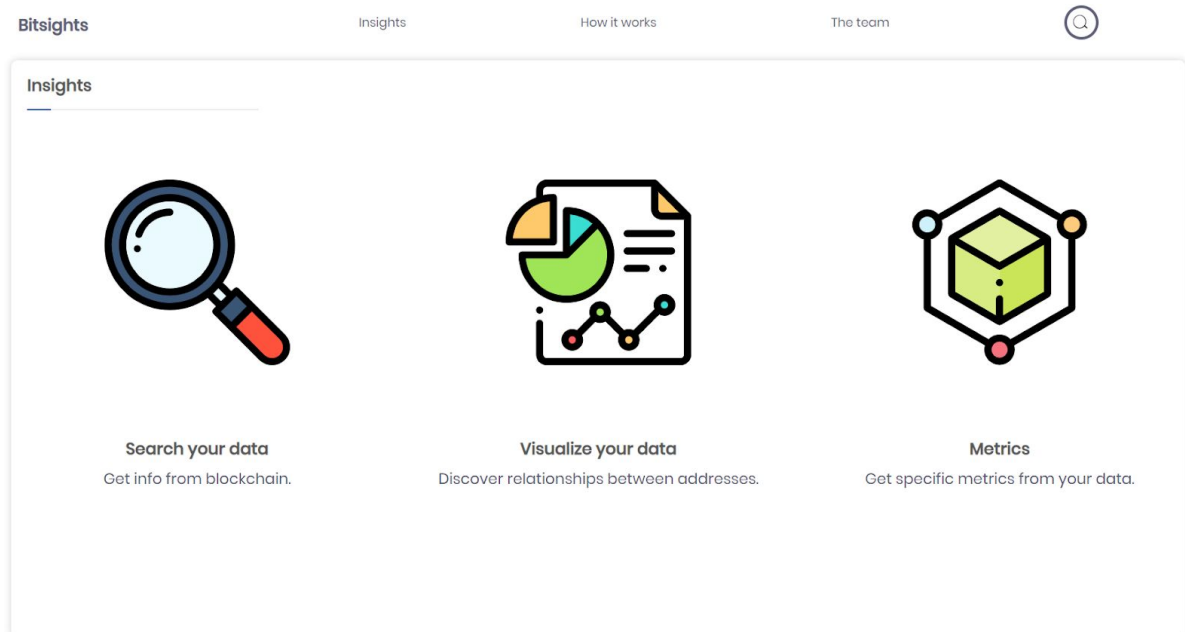
Image 20 - Metrics of an address are shown when clicking on the Heuristics button.

Demo - example of usage

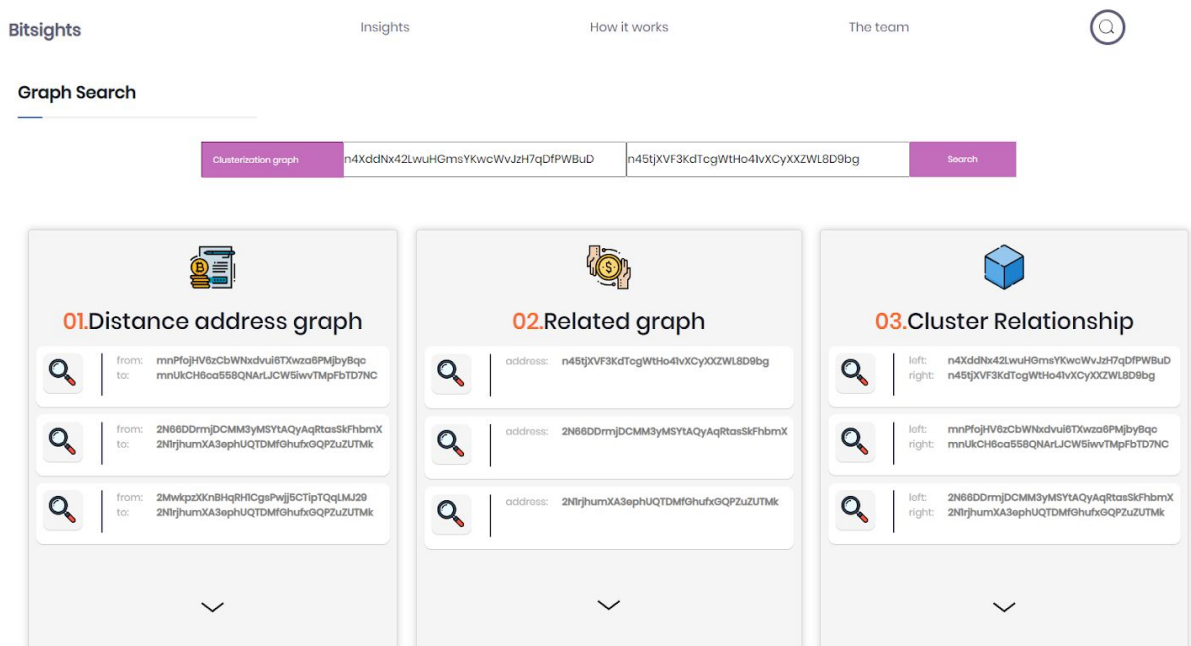
Using the web application, steps needed to be applied in order to get those results are listed below. First of all, “Insights” section was chosen and several options are presented:

- Search data: here, users can search data associated with addresses, transactions or blocks.
- Visualize data: this section delivers a graphical representation tool helping users to have a clear portrayal of its data. Nowadays, graph variation is firmly correlated to the heuristics previously exposed, which means that there are three types of graph available (Distance transaction, Related and Clusterization graph).


- Metrics: users can take advantage of it by searching different values that holds on an address: wallet probability, cluster balance, time series balance and volume value.



After clicking on “Visualize your data” section, a new interface will be shipped offering users the possibility of searching its desired content or choosing from addresses previously searched with known results.



So, whenever any of those possibilities have been chosen, after clicking on search button two things can happen: result was successful and user can click on “Details” button inside successful message, or no results could be found due to any issue related to typos when typing BTC addresses or because of a failure in the api provider used, which is in this case bcoin.

[Bitsights](#)
[Insights](#)
[How it works](#)
[The team](#)


Graph Search

Clustering graph

n4XddNx42LwuHGmsYKwcWvJzh7qDfPWbuD


n45tjXVF3KdTcgWtho4lvXCyXXZWL8D9bg


Search


Search results


Address from
n4XddNx42LwuHGmsYKwcWvJzh7qDfPWbuD


Address to
n45tjXVF3KdTcgWtho4lvXCyXXZWL8D9bg


Details




01.Distance address graph



from:
mnPfojIV6zCbWNxdvul6TXwza8PMjby8qo



to:
mnUkCH8oa558QNaRJCW5lsvTMPfTD7NC



02.Related graph


address:
n45tjXVF3KdTcgWtho4lvXCyXXZWL8D9bg


address:
2N86DDmJDCMM3yMSYIAQyAqRtas5kFhbmX

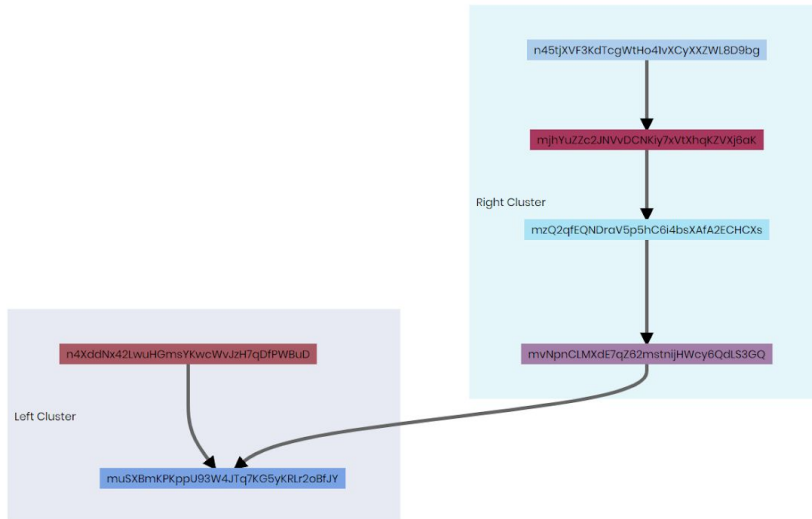

03.Cluster Relationship


left:
n4XddNx42LwuHGmsYKwcWvJzh7qDfPWbuD


right:
n45tjXVF3KdTcgWtho4lvXCyXXZWL8D9bg

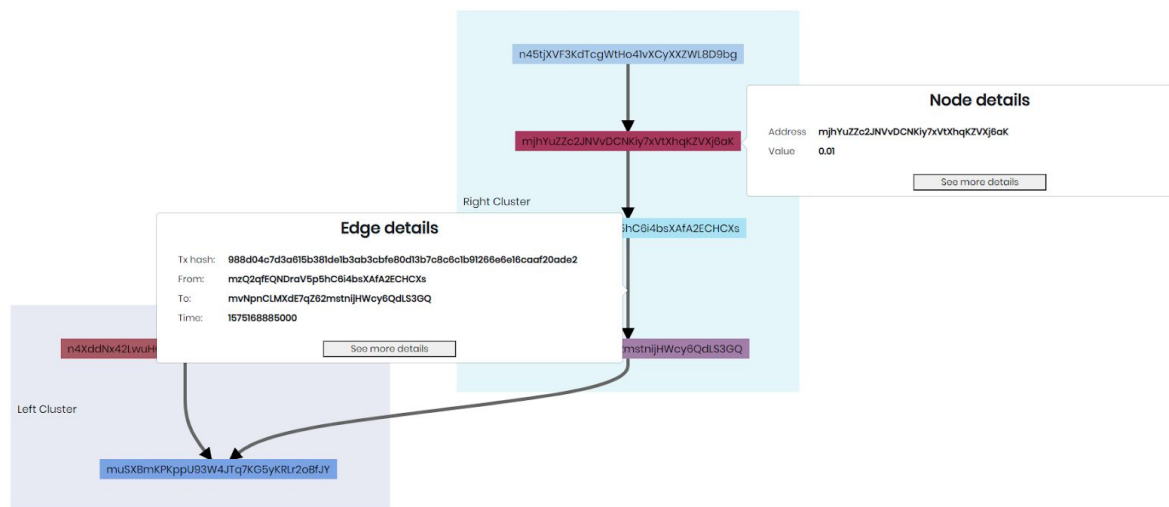
After clicking on the Details button, a new interface will be deployed showing three clear and distinct tabs: “Graph”, “Details” and “Download”. First and default option shows the graph obtained from applying the corresponding heuristic.

Address clusterization graph



Something to point out is that graph visualization is not static, meaning users are allowed to interact with it. Details to be taken into account: nodes can be moved from one place to another, nodes and edges can be hovered bringing information about clicked element (this is helpful when dealing with huge graphs and its quite hard distinguishing elements). Last but not least, node/edge details disappear when users click anywhere in the page and also can access to see details by clicking on the “See more details” button.

Address clusterization graph



By clicking on the “Details” tab, users have access to different data associated to graph: nodes number, edges number, change addresses. max value nodes, cluster data. Besides, when clicking on any of the nodes or edges listed in the “Nodes/Edges” panel, a

“Node details” or “Edge details” section will be displayed, showing more accurate information regarding the selected element and a little portrayal of its selected element.

Graph Details Download

Graph details Cluster 1 Cluster 2 Cross edges

General cluster metrics

Nodes number:	2
Edges number:	1
Change address:	-
Max value node:	0.00999434

Nodes Edges

n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD
mu5XBmKppl93W4Jtq7K05yKlr2oBfJY

Node details

Address n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD

Connected to: mu5XBmKppl93W4Jtq7K05yKlr2oBfJY

Click to see more details >

Representation in graph

n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD

Value: Show Heuristics BTC

Depending on the option selected, “Click to see more details” will load details associated with node address or transaction hash edge.

Graph Details Download

Graph details Cluster 1 Cluster 2 Cross edges

General cluster metrics

Nodes number:	2
Edges number:	1
Change address:	-
Max value node:	0.00999434

Nodes Edges

n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD
mu5XBmKppl93W4Jtq7K05yKlr2oBfJY

Node details

Address n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD

Connected to: mu5XBmKppl93W4Jtq7K05yKlr2oBfJY

Click to see more details >

address	n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD
balance	0
final_balance	0
final_n_tx	2
n_tx	2
total_received	1000000
total_sent	1000000
unconfirmed_balance	0
unconfirmed_n_tx	0

Representation in graph

n4Xd8Nx42LwuhGmsYKwcWvJzH7qDfPWBuD

Value: Show Heuristics BTC

The edge details section shows inputs/outputs involved in the transaction, as well more data can be obtained by clicking on the “Click to see more details” button.

Edge details

Hash: 6eef6df1e74d66932100081ad3400f76dfadefee8e6cfc802dc1b62d248011a5

Sun Oct 06 2019 06:07:50 GMT-0300 (Argentina Standard Time)

Inputs

0.00999434 BTC

mu5XBmkPKppU93W4Jtq7K05yKRLr2oBfJY

0.01 BTC

n4XddNx42LwuHGmsYKwcWvJzH7qDPFWBuD

Outputs

0.0099908 BTC

mg8ausGuQJ9XQ3Nh92VcJfJhEW5yIvrvf

0.01 BTC

mnUkCH6ca558QNArLJCW5iwwTmPfbTD7NC

Representation in graph

n4XddNx42LwuHGmsYKwcWvJzH7qDPFWBuD

Value: BTC

Show Heuristics

mu5XBmkPKppU93W4Jtq7K05yKRLr2oBfJY

Value: 0.00999434 BTC

Show Heuristics

Click to see more details >

Edge details

Hash: 6eef6df1e74d66932100081ad3400f76dfadefee8e6cfc802dc1b62d248011a5

Sun Oct 06 2019 06:07:50 GMT-0300 (Argentina Standard Time)

Inputs

0.00999434 BTC

mu5XBmkPKppU93W4Jtq7K05yKRLr2oBfJY

0.01 BTC

n4XddNx42LwuHGmsYKwcWvJzH7qDPFWBuD

Outputs

0.0099908 BTC

mg8ausGuQJ9XQ3Nh92VcJfJhEW5yIvrvf

0.01 BTC

mnUkCH6ca558QNArLJCW5iwwTmPfbTD7NC

Representation in graph

n4XddNx42LwuHGmsYKwcWvJzH7qDPFWBuD

Value: BTC

Show Heuristics

mu5XBmkPKppU93W4Jtq7K05yKRLr2oBfJY

Value: 0.00999434 BTC

Show Heuristics

Click to see more details >

block_hash

"0000000035070a74381c69056cd5974cc19d88662d39c9ac980566f66d0a38b6"

block_height

1580587

block_index

103

confidence

1

confirmations

199814

confirmed

"2019-10-06T20:07:50Z"

double_spend

false

fees

374

hash

"6eef6df1e74d66932100081ad3400f76dfadefee8e6cfc802dc1b62d248011a5"

lock_time

1580586

opt_in_rbf

true

preference

"low"

relayed_by

"54.146.19.151:8333"

size

374

total

1999060

ver

2

vin_sz

2

vout_sz

2

One more aspect to be considered is the possibility for users to get metrics instantly by clicking on the “Show heuristics” button in the “Representation in graph” section.

Cluster balance -0.000013920000000002333 BTC

Wallet probability -0.000013920000000002333 BTC

Volume Inbound 0.04998676 BTC

Volume Outbound -0.03998868 BTC

Balance time series graph



Last section is “Download”, which performs a GraphViz representation of the graph. One peculiarity that cannot be obvious is the fact that users are able to download this representation as a .png file. This simple feature is extremely powerful, because users can obtain its data and then keep it, acquiring a snapshot of it through time.



Applications

Applications for this work are very specialized, and focused around deanonymization of blockchain information. The most straightforward application concerns address clusterization, where given an address, the user is able to find every other address that the same owner may possess, and to thus discover the full wallet.

Another application concerns fraud detection, in the context where if an address is known to have participated in a fraudulent scheme, it is possible to find other victims, or other addresses that have participated. For example, if you know the address of a scammer requesting payment for a given reason, you are able to find other addresses that the scammer may own. As this system allows an exchange, or an interested party, to find transactions not directly related to an individual. It is possible to analyse the full transaction history of a wallet, extract usage patterns, balances, and other related information.

All this information is useful to profile and score a set of wallets, and then rate how likely an operation may result from fraud.

It is also possible to use this system to find relationships between individuals, in this case in the form of transactions. This information may be used by law enforcement, or even social networks and advertisers.

Additionally, it allows the augmentation of the information already available in the blockchain. This can be used to perform further studies related to clusterization, or financial analysis. One such application could be preventing money laundering by finding previously unknown transactions, or the usage of a service for this exact purpose.

Limitations and future work

The biggest limitation found during this research is the sheer size of the data that needs to be locally available for this system to work. This was worked around by using a Blockchain Explorer that allowed us access to their API, but for a full deployment of this service, a full node would be much more fast and efficient.

As was previously mentioned, the minimum amount of data that needs to be persisted according to our estimation is 750 GB. And this is without taking into account the time needed to process the full blockchain, and the extra space required to store the indices.

From one of the papers found during the research phase, a full-chain analysis was performed in November of 2013 using a cloud VM with 26 virtual CPU cores, and 68 GB of

RAM memory. With these specifications, the full blockchain could be loaded in memory, and the analysis took a full 45 minutes to complete. It goes without saying that with the exponential growth that Bitcoin had since then made this problem much worse.

Given these limitations, the use of graph databases like Neo4j was quickly discarded, as space and system requirements are much more demanding for this kind of system. The first improvement the team could propose, would be the use of a preprocessing stage to transform the raw blockchain into a structured graph, and thus allowing the use of queries that an engine like Neo4j supports. This would reduce the address distance heuristic from a BFS search, to the application of the Dijkstra algorithm. Although the use of a relational database engine was tested, it was quickly discarded because of the same reasons.

One final test that was impossible to finish in a reasonable amount of time, is to analyse how the address clusterization heuristics perform by scanning the full blockchain, and calculating how many clusters are found, and what percentage of the address space can be found in a cluster larger than an arbitrary number.

Regarding graph implementation, it is important to mention the fact that the library does not offer more than the possibility of showing graphs and when trying to do something like choosing subgraphs or choosing a group of nodes/edges for a further analysis is not natively developed. Instead, those details require restructuring code and overthink graph templates. Despite that, it is worthy to mention that as how the application is built, it is possible to add future features easily. Two possibilities were thought of as later projects like giving the chance to users to create their own graphs (it implies template graph customization as well as creating nodes with desired format) in order to make a more manual analysis/investigation and allow them to make comparisons between graphs.

Conclusions

Blockchain analysis proved to be a real challenge. As these technologies reach mainstream usage, and their adoption grows, so does the amount of data that needs to be stored and analysed.

One of the foundations of the cryptocurrencies is their openness and anonymity. All of them are developed in Open Source projects, and auditable by the community. But, with the challenges presented during the evolution of this paper, it was evident that access to the development of this kind of services is being increasingly limited to economically interested parties, with the capabilities of upfronting the cost of performing such analysis.

This being said, the first conclusion reached during the development of this paper is that the difficulty to develop software to perform analysis of the blockchain, and to later provide a service based on it, is ever increasing, and restricting access to the mainstream cryptocurrencies as they continue to grow both in size and in volume.

A lot of research had to be done around the structure of the Blockchain, and how it can be traversed. At the same time, some significance had to be given to all this information, so that it can be used to enrich, and link all the related pieces. From studying the binary format of the blocks, to running a full node, a full overview of all the available alternatives to access this data set was needed.

Working around the limitations, several heuristics that allowed the enrichment of the information stored in the Bitcoin Blockchain, and an MVP of a website used to query and visualize this information were developed.

During this process, various approaches were investigated, and promptly discarded due to their inability to scale. From importing the blockchain transactions into a relational or graph database, to raw parsing of the block information. Thanks to access given by a third-party API that allows queries to be performed on the mainnet, some interesting results are shown to give an example of the information that the implemented heuristics can generate.

The biggest challenge that was faced, was the development of a website able to show the information that the system that was developed was extracting. A lot of work had to go into testing and validating the best ways to represent large amounts of related information as graphs, and not overwhelm the user, while at the same time making this visualization easy to browse. As a result, it is now possible to present the user big amounts of information regarding addresses and the transactions that link them, and provide some insights into the transaction habits, or accumulated wealth of a wallet, among other things.

Although the project started with the goal of completely deanonymizing a given set of addresses, it was quickly proven that this task would be impossible without an anchor, or some piece of information to link an address belonging to a cluster to the real world. This would of course depend on the development and integrations of other sources of information, such as web scraping, or querying search engines.

Finally, and disregarding these limitations, some enhancements were proposed to provide a faster experience for the execution of the heuristics developed.

References

- <https://fc17.ifca.ai/bitcoin/papers/bitcoin17-final11.pdf>
- <http://graphsense.info/>
- <https://medium.facilelogin.com/pay-with-bitcoin-to-play-with-a-fidget-spinner-86b7b43414c0>
- <http://codesuppository.blogspot.com.ar/2014/01/how-to-parse-bitcoin-blockchain.html>
- <http://www.syssec.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/system-security-group-dam/research/publications/pub2012/596.pdf>
- <https://nickler.ninja/papers/thesis.pdf>
- https://www.ifca.ai/fc14/papers/fc14_submission_11.pdf

Annex

Web application Implementation

Until this point, there is a backend solid structure which consumes resources in order to handle the mainnet and testnet, collects all data, uses it in data structures, filters it by using coding strategies and algorithms and, with all that set, calculates a variety of heuristics and returns data associated with clusters relationships, distance from a source to a sink address, and the importance of finding change addresses. But it is a must to show this data in a feasible way. By referring feasible, it is assumed that inputs need to be accessible simply, quickly, and effortlessly.

Taking everything into account, a frontend was developed to fulfill these objectives:

- Everyone can use the website, no matter what kind of bitcoin technical background an user has.
- Every piece of information that is shown has its purpose.
- Data can be accessible in different ways.
- Managing a big amount of data and making it understandable.
- It should be easy arriving to a conclusion or, at least, states possibilities/paths never thought or imagined by an user.

Through these pages, a further explanation of Bitsights tool will be done, highlighting topics associated with the previous investigation before materializing the idea, its construction process, issues that appeared while progressing, results obtained, conclusions and possible/future improvements to it.

Process

Creating a web application is utterly distant from building a contact web site or a blog. Although both share several stages regarding designing process or testing, a significant difference is that in this case a visual tool specifies an abstraction of an idea and reveals it to the public as a concrete product. In this case of a study, Bitsights is the culmination of different phases.

Materializing the idea

Before starting digging in how to structure the application and coding it, defining which kind of application was going to be assembled was a critical step. Assuming that there is a lot of information that needs to be displayed somehow, developing a command-line interface was not an option. Nowadays, there are a variety of alternatives that can replace an old-fashioned shell. Despite it still exists, it has a lot of usages and is applicable in many cases such as virtualization tools, container, or cluster orchestration, writing commands or scripts and getting a plain text response, it does not achieve the purpose of presenting data in the most significant form.

Taking advantage of internet and telecommunications, the selected candidates seem to be a web, desktop or a mobile application. Which sounds logical due to the fact that it is more plausible everyone knows how to interact through a desktop/web application, or a mobile phone. Although it looks like building a mobile application would be easy after constructing a webapp or a desktop one, it is not. It implies more resources, investing more time to have several products in production, more maintenance because if an issue arises, two points of failure are now into consideration. More testing, more technologies, just a simple list of more and more.

Moreover, when thinking about how to show different cluster relationships or huge paths between two addresses or just simple metrics, the context suggests considering more sizable screens or, at least, bigger sizes as the one an user finds in a cell phone.

Last but not least, a desktop application does not apply to this example because considering that option means that customers ought to have experience with downloading

tools. Another disadvantage is bringing support for every platform (Windows, MacOS and Linux, with its updates) and different versions.

That is why the decision of building a web application as the solution for this study case was taken. The client opens a web browser (of course there are a variety of choices and flavours, but there is a stable web standard and no impossible bottlenecks to deal with), enter the URL and that's all, then only thing is navigate through the site and have fun.

Designing process

One of the biggest challenges this application had to elude was the representation of data, not only the path between addresses, but also the details view for a specific piece of information. Basically, the problems to solve can be divided as these questions:

- How to represent addresses, transactions and clusters on a website?
- What information should be presented when requesting for a specific transaction ID, address ID, block ID?
- Should users interact with data dynamically or just show content statically?

To begin with, and as it was previously explained, a transaction is a transfer of Bitcoin value that is collected (or mined) into a block. It has an identification number, known as txID and is represented by 32 bytes (64 characters) and hexadecimal. Besides, an address is also represented by a hash value and its length is shorter, just 20 bytes. Having that in mind, a transaction between two members can be visualized as a path (source address - transaction -> sink address). What structure is best to represent connections or paths between members? A graph, which consists of nodes (addresses) and edges (transactions). Graph structure fits perfectly to represent the data. When considering showing it to users, several things need to be considered.

In the easiest case, a trivial path consists of two addresses and one transaction:



Image 21 - trivial path formed by two addresses and a transaction

Or two addresses with two transactions (considering third address as a change address):

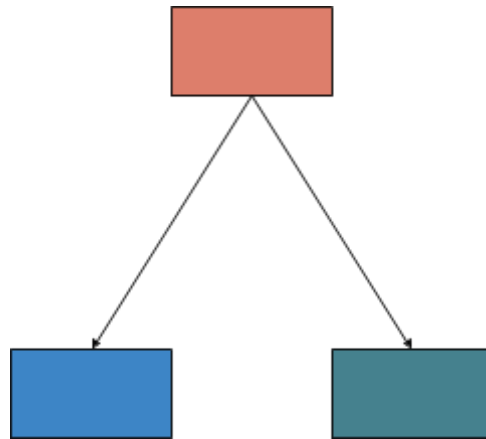


Image 22 - complex path formed by three addresses and two transactions

It seems affordable to achieve it with no complications. But, in both cases the user does not know what node is representing which address. Same happens with transaction hashes. These scenarios serve as an example:

Case 1)

Address source: 16Fg2yjrwtC6fZp61EV9mNVKmwCzGasw5

Address sink: 1LCAJF94Yxin9eWNx19b5BZnrnBSVstV1g

Transaction: 9addc82c0cfbfe2e53fd3de2c437d12b3b0f3bd8a1fed4b2162128e1a9b7755a



Case 2)

Address source: 346LuU4QtuAqPsYxPjGjt8RkSr2p4vKjdF

Address sink1: 16Fg2yjrwtC6fZp61EV9mNVKmwCzGasw5

Address sink2: 38pbnmt8kM6hPiFziyjCHI3uRqVTCRMM55

Transaction1: 9c837ab70703fa0b23a5e7982f0b98f453ca6c9081a4cf6bb8fa641ef4f293e3

Transaction2: 56f5211507cc807540a9d392be9ad2c517783f3b8d316547c813df1ba73b4948



In both graphs, issues to point out are:

- absence of clarity for an user to distinguish addresses and hashes because of font-size
- difficult label positioning for hashes
- difficulty with nodes and edges positioning as graph size grows

Another topic associated with is the graph orientation, which impacts directly in web app navigation. As data grows, huge datasets must be represented in an efficient way, taking advantage of the screen. So, an experiment was done to decide either horizontal or vertical direction. Continuing with previous parameters (as previous image of Case 2 can be used as an example of vertical direction, horizontal direction is included):

Horizontal Direction



Image 23 - horizontal representation of a simple graph

In order to have a more accurate test case, more nodes and edges were added to the graph.

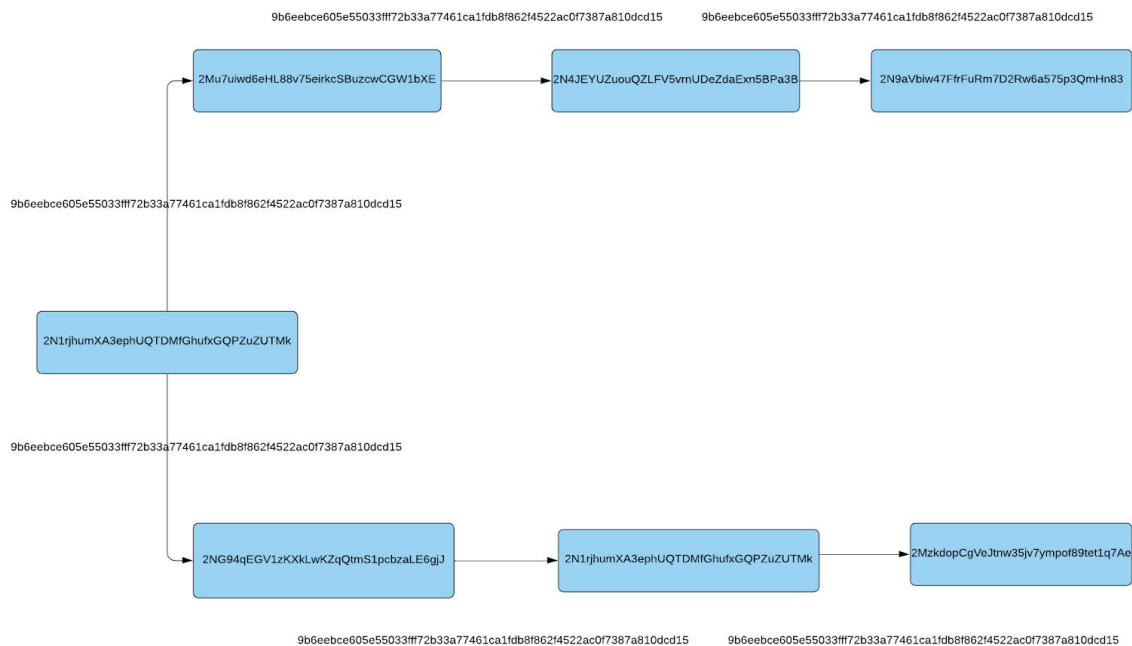


Image 24 - horizontal representation of a complex graph

This is what it looks like when the graph has more components. It gets confusing because there is no clear hierarchy. It is hard to recognize where the graph starts and ends. When the amount of nodes and relationships grow, users need a clear form of distinguishing at least the beginning of a graph to get an idea how deep is. With this set up, the usage of horizontal bars in order to see the full graph is not enough. As an alternative, managing components the other way looks like:



Image 25 - vertical representation of a complex graph

Despite addresses font size being small and transaction edges not clear enough, vertical disposition presents an improved form of presenting data. Regarding web design, if a path between two addresses becomes larger, it will be reflected in an increase of the web page vertical height, which means the user needs to scroll down to get to latter nodes. It could be tedious, but nowadays there are several applications that expose its content in only one page, and if users want to go to deeper sections, they are compelled to scroll down. Moreover and as it was previously explained, vertical content disposition helps users to have an accurate location of the graph. Having explained that, the decision was made on presenting data using vertical disposition.

Now it is time to adjust other points to make this application more coherent:

- tiny font applied to addresses and transaction labels which adds more complexity to graph comprehension.
- adjustments in transaction labels not matching edges.

Both points are directly connected to screen sizes. Since the choice is a web application, several display sizes must be taken into account. From 13" inches to an extent variety of multiple proportions and dimensions. Though, another outcome rises: with the growth of technology, it is usual to visit a web page via mobile phone. So, another problem is presented: how will the application handle this variety? Fortunately, there are solutions that allows to solve this problem generically.

Applying concepts of responsive web design solves this problem: developing a responsive web app ends with this dimensions issues. Likewise, it is affordable to apply because it only implies specifying CSS rules for a group of common sizes that applies to the vast majority of mobile devices and notebooks.

Finally, an idea was applied to avoid transaction label adjustments or minuscule addresses hard to understand: the usage of tooltips. A tooltip is a way of showing additional data related to something. Generally, it is triggered when the user moves the mouse pointer over the element that holds this extra data. Applied to this case, an element could be a node or an edge. So, when these elements are hovered, the tooltip would display its details, reducing the extra amount of space needed as shown in previous cases and allowing the user to distinguish data and focusing on the right thing.

According to that, two examples are exposed next, showing how this interaction works for these pair of addresses and transactions:

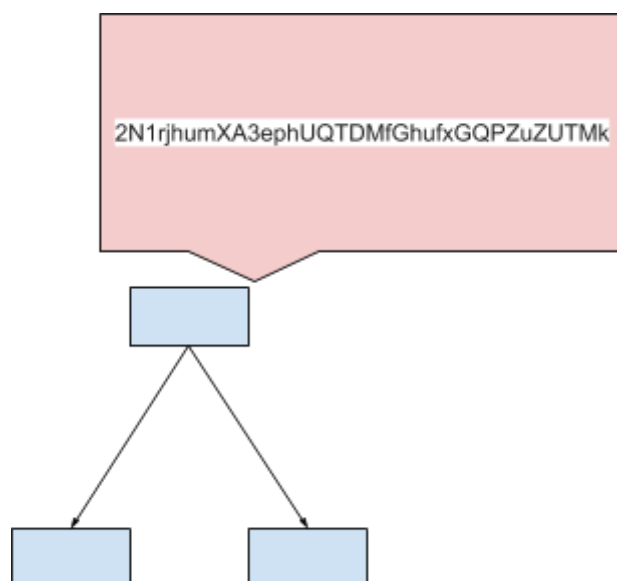


Image 26 - representation of a tooltip when hovering a node

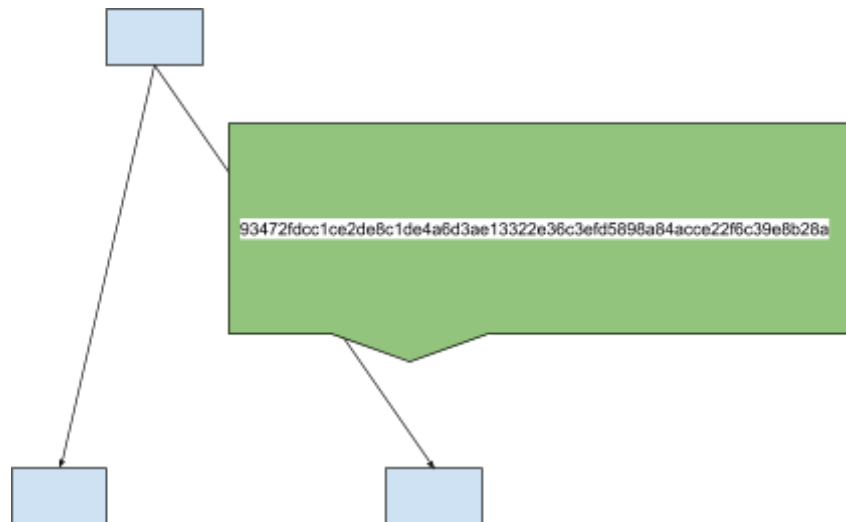


Image 27 - representation of a tooltip when hovering an edge

Now, not only is clarity improved, but also it is settled that content should be presented dynamically, which means the user has to interact with the graph to get details or specialized data.

Another subject that was previously mentioned but never detailed is cluster depiction. As it was said, a cluster is composed of several addresses, possibly belonging to the same individual. Graphically, it would be a group of nodes connected with edges. But to really make it clear that these elements are part of a group/category, a representation must be applied. The most accurate one is a bag or box that holds elements.

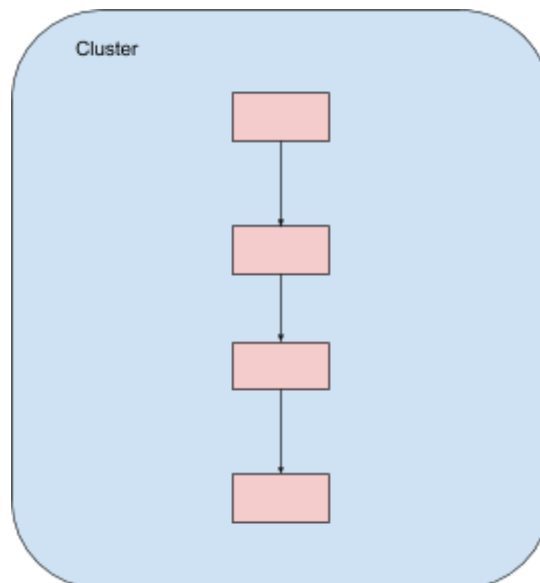


Image 28 - representation of a cluster

So, the next step was to define how to integrate this subgraph in Bitsights. An earlier explanation when using “Cluster Relationships” heuristic invites to think how to display two related clusters of addresses with its edges that cross boundaries between them. Fortunately, applying preceding reasonings, the solution is straightforward: show two boxes, each one with its addresses and edges in vertical dispositioning, as well as cross edges.

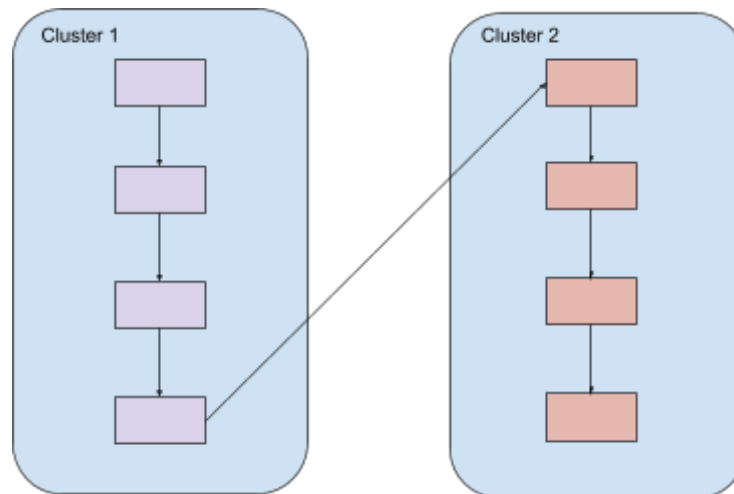


Image 29 - representation of an interaction between clusters

Last but not least, a hot topic for a dedicated analysis was what information should be shown when requesting a specific address, block or transaction, likewise how to connect this within a graph representation. As it was mentioned before, the purpose of Bitsights is representing filtered data, so users have to focus only on arriving to conclusions or make further analysis with that. The next paragraphs try to explain how to represent:

- block details
- transaction details
- address details
- graph details

As a first step, the idea of presenting blocks was associated with a timeline as a simulation of a Blockchain in which every milestone would be a block. So each time a milestone was clicked, a thorough interpretation of a block appears. This will be explained later in the implementation section.

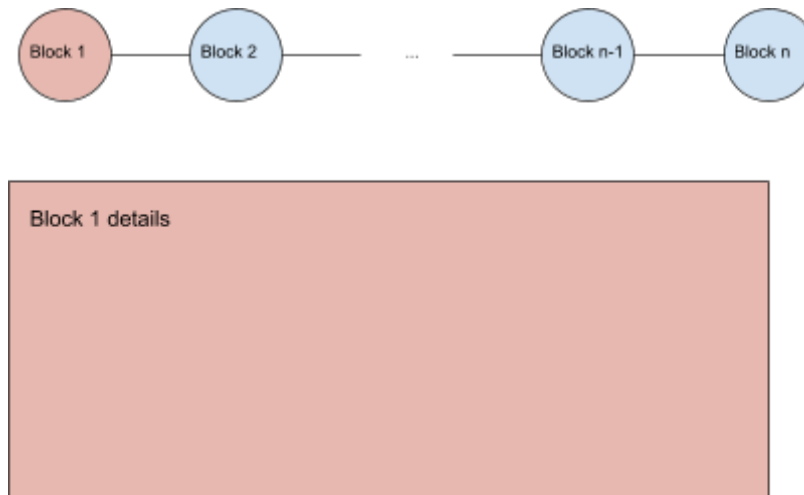


Image 30 - representation of a blockchain using an horizontal timeline in which each of its elements is a block.

Furthermore, a block contains transactions. In fact, they were imagined as part of the block details section using a vertical list, in which every list member consists of a rectangular shape showing a transaction hash and when clicked, its height increases revealing all details. Whenever you clicked on another list member, this previous rectangle should be hidden, allowing the appearance of the details connected to recent clicked member.

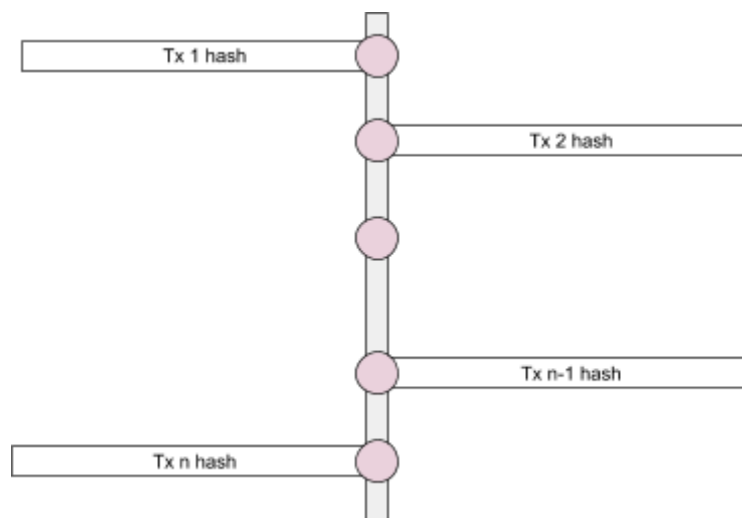


Image 31 - representation of a list of transactions using a vertical timeline.

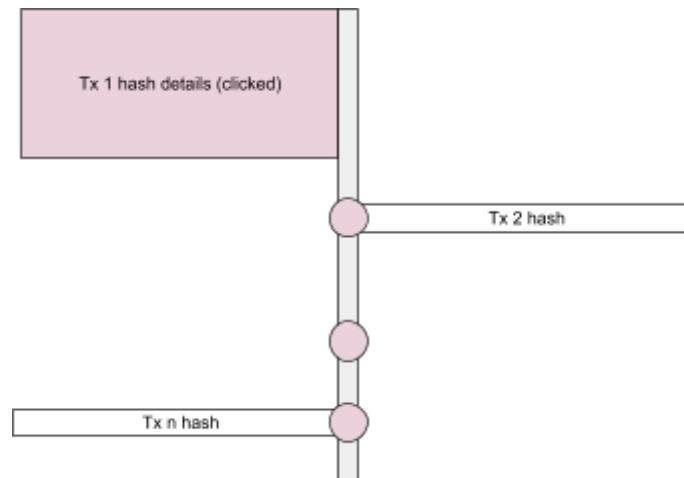


Image 32 - An element of the transaction list being selected, showing its details.

Same strategy was applied to transaction details, when presenting inputs and outputs. In this particular case, two vertical lists were used: one for inputs and the other one for outputs.

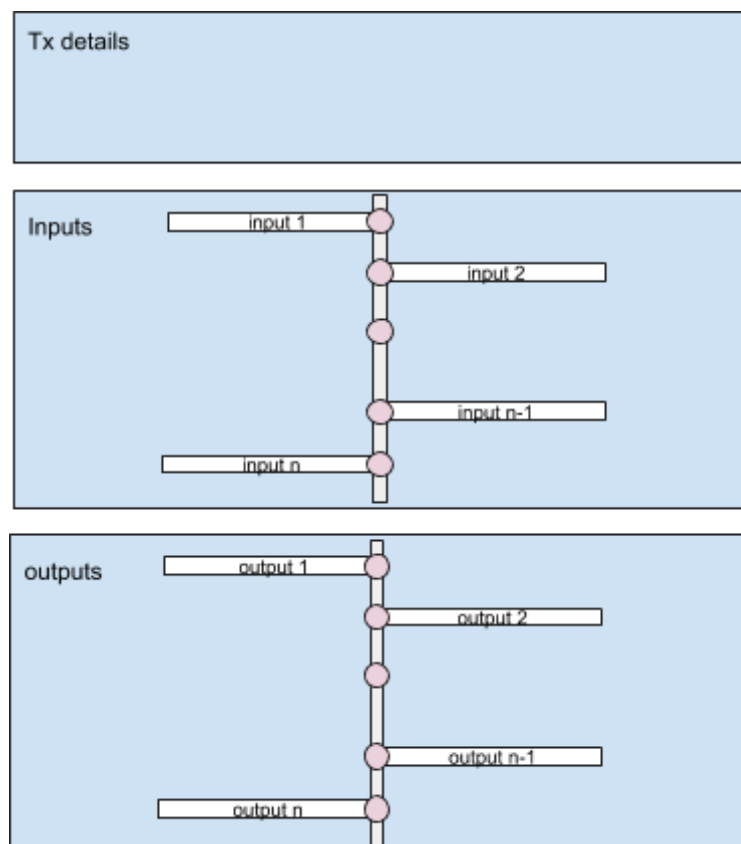


Image 33 - representation of inputs and outputs lists, using the same schema as transactions.

In fact, it was established using this layout for graph details too, in order to have a coherent navigation through web pages. In case of distance and related graph, three sections were included: details, nodes, and edges, all following the same structure. On the other hand, for clusterization graphs, another pattern should be followed because of the subgraphs: each one contains its group of nodes and edges. So, two buttons were added in order to change between a cluster or another and avoid content repetition, putting the same sections one after another.

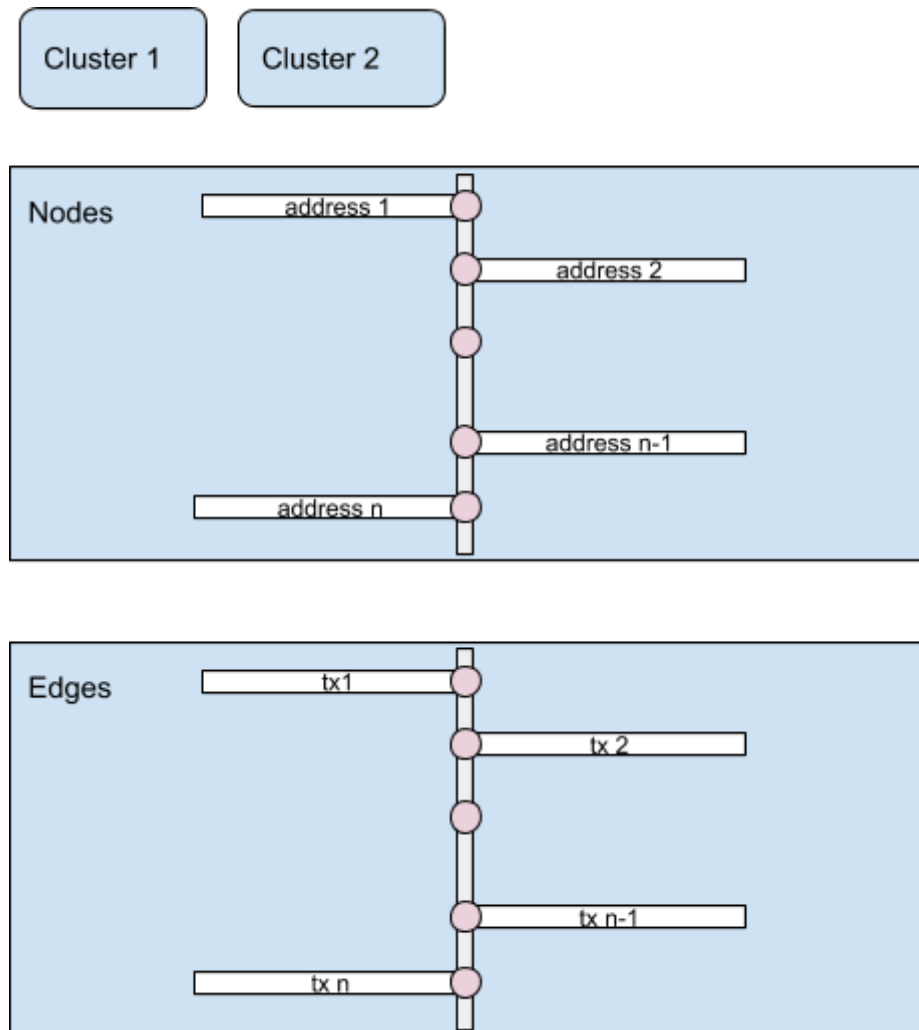


Image 34 - representation of cluster details section, including nodes and edges using a vertical timeline approach.

All this background served as an input for the implementation, based on the takeaways that came up after designing process: develop a web application, letting users search data associated with addresses, transactions, and blocks and allowing them to get a graph visualization on certain insights, previously defined as heuristics, in a graph representation that consists of nodes and edges (inside a cluster representation when

needed) displayed in a vertical orientation for a better understanding, and that uses designing strategies such as tooltips or timeline.

Implementation

The implementation of the web application proved to be a challenge due to the lack of experience of the team when building frontend projects. Bitsights was written using Angular, a framework used for creating single-page apps. More in depth, the project uses Angular version 6. It is important to mention the fact that two versions of the app were made because of design complexities that arise while building the first approach. Details on the motives, how these points were solved and results will also be mentioned.

Skeleton structure

For both implementations, code is based on this architecture: there is a starting point “app component” that holds references to the rest of components, services and external libraries.

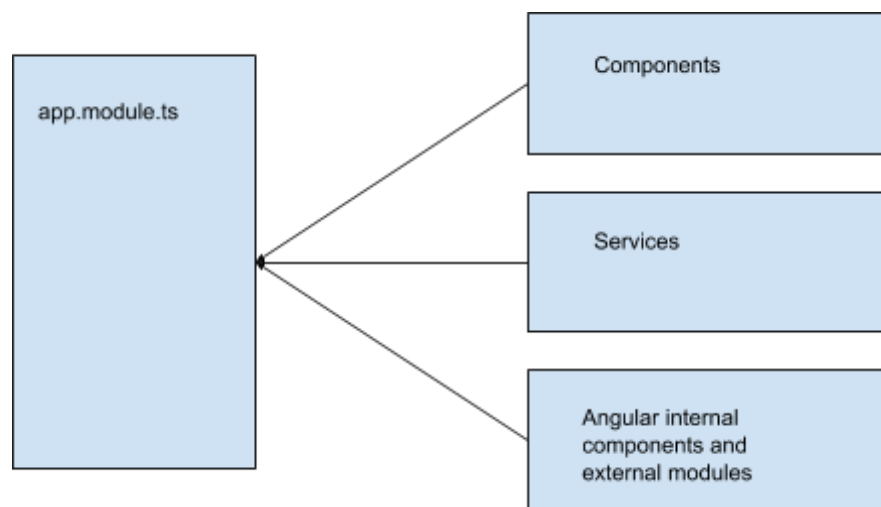


Image 35 - skeleton structure used in development of web application.

The content orchestration when navigating through different pages is in charge of the app-router-outlet, which interprets changes in the browser URL and shows different page views.

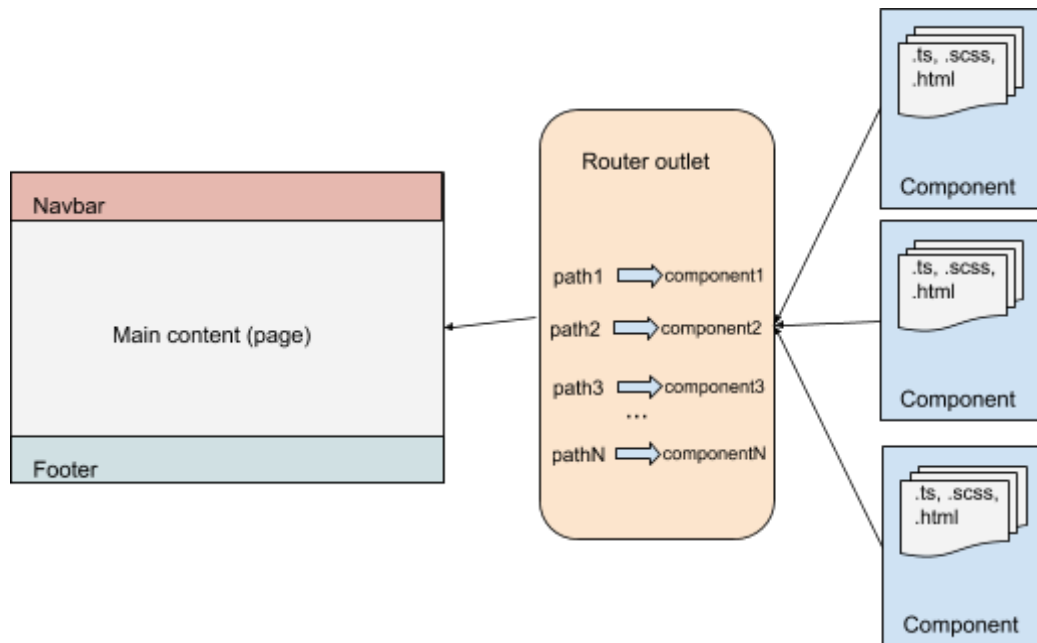


Image 36 - A brief portrayal of router navigation flow in Angular.

Components that are part of Bitsights, which then will be pages shown to users, are listed here:

- Home: it is the main page where the tool is presented to users.
- HowItWorks: basically, it is the application documentation. It explains in detail the three basic ways of using the tool, which are: search addresses, transactions or blocks; visualize different kinds of graphs: related, distance and clusterization graph; obtain more metrics.
- Team: It is the contact page which presents members of the team and its background.
- Insights: a simple page that offers which option users want to select
- Search: allows users to search addresses, transactions or blocks. In case someone doesn't know what to search, there is a section with suggested terms.
- BlockDetails: it shows info related to the searched block. And also it shows its tx.
- TransactionDetails: Not only displays tx details, but also its inputs and outputs.
- AddressDetails: it exhibits info associated with the searched address.
- Graph: this page is similar to 'Search', with the slight difference that offers suggested terms for looking into graphs.

- **GraphResults:** In conjunction with details, this page is the most important one. It combines three tabs, where users can see: a graph visualization of searched terms, details of it and the possibility to download the content as an image.
- **Data:** Similar to 'Search' and 'Graph', it offers suggested terms for users in order to search heuristics, splitted in four categories: time series balance, cluster balance, wallet balance and volume balance.

When communicating with the backend, it was necessary the creation of a tiny set of services, which work as helpers and its main purpose is to pre-process data given by the API service, which later will be shown to users. These services are:

- *Address Service:* calls API service to get a specific address and is in charge of bringing an array of transactions associated with a searched address.
- *Transaction Service:* calls API service to get a specific transaction, as well as its inputs and outputs.
- *Block Service:* calls API service to get a specific block and saves the latest searched result to avoid calling the API many times when showing details needed.
- *Graph Service:* calls API service to get data connected to different kinds of graphs, mentioned in previous sections. Saves latest search result to avoid calling the API many times when showing details needed. Furthermore, it brings info that has a lot in common with heuristics.
- *Details Service:* as its name describes, is in charge of preprocessing results saved in any of the previous services and preparing a more readable data, then consumed by the views.
- *HistoryService:* Its main objective is maintain the app flow when getting back to a visited page and see previous searched results.

Main flow

Searching data is one of the main functionalities Bitsights offers, basically because of the variety of filters users are allowed to select. Basically, this monolithic thread fulfills same pattern:

1. An action is triggered.

2. A request is performed to the server. Depending on the case, the request will hit a Blockchain provider server or it will create a job on the server internally, getting data requested and preprocessing it.
3. Data is obtained from the server and is shown.

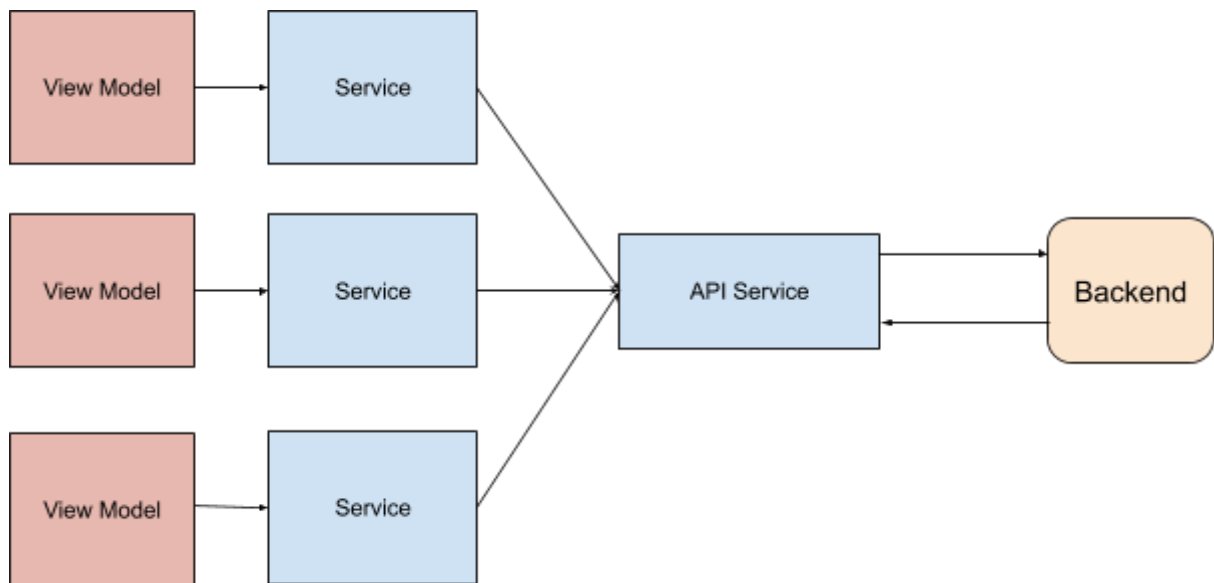


Image 37 - Backend communication flow chart.

Details pages

With those sketches prepared, previously mentioned in the designing process and showed in pictures, the next step was to implement it.

First implementation

In this case two libraries were selected: `angular-mgl-timeline` and `shalugin/horizontal-timeline`. The first was used to show tx references but it was discarded due to its complexity to show huge content. The second one was applied to the blockchain view and it was dismissed due to some bugs when rendering elements when loading several gb of blockchain data.

Second implementation

The idea was to rethink web pages in a way that seems easy to understand for users, and of course, easy to develop. These new visual changes contemplate intuitive navigation and interaction with elements displayed as well as the intention of showing big amount of data in a clear way. Mainly, modifications included: a) the addition of sections to

distinguish which info is associated with which element (for example, separate address properties and values from tx references), b) the usage of line separators, helping users not feeling overwhelmed with huge content, c) utilization of buttons that helps to reuse sections but change its content when triggered.

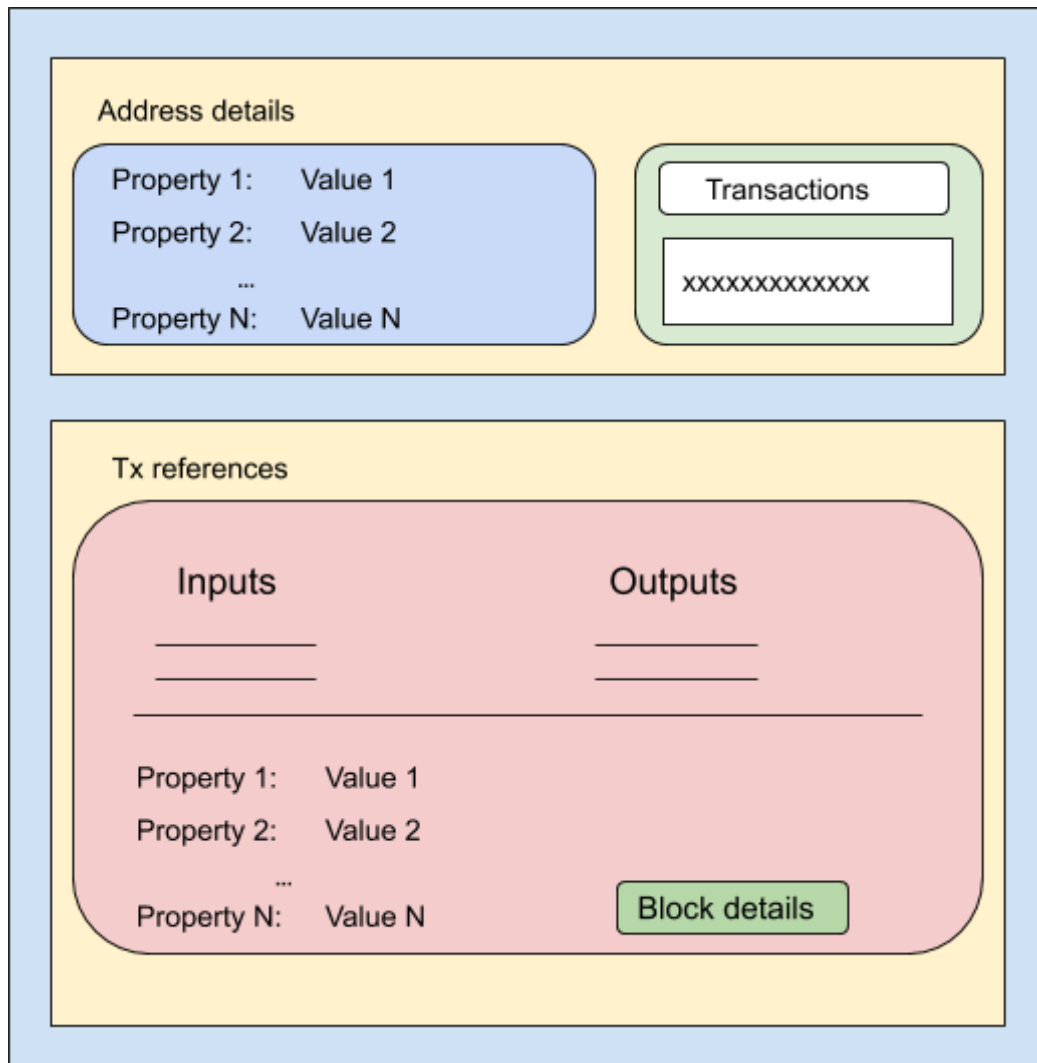


Image 38 - Address details new layout.

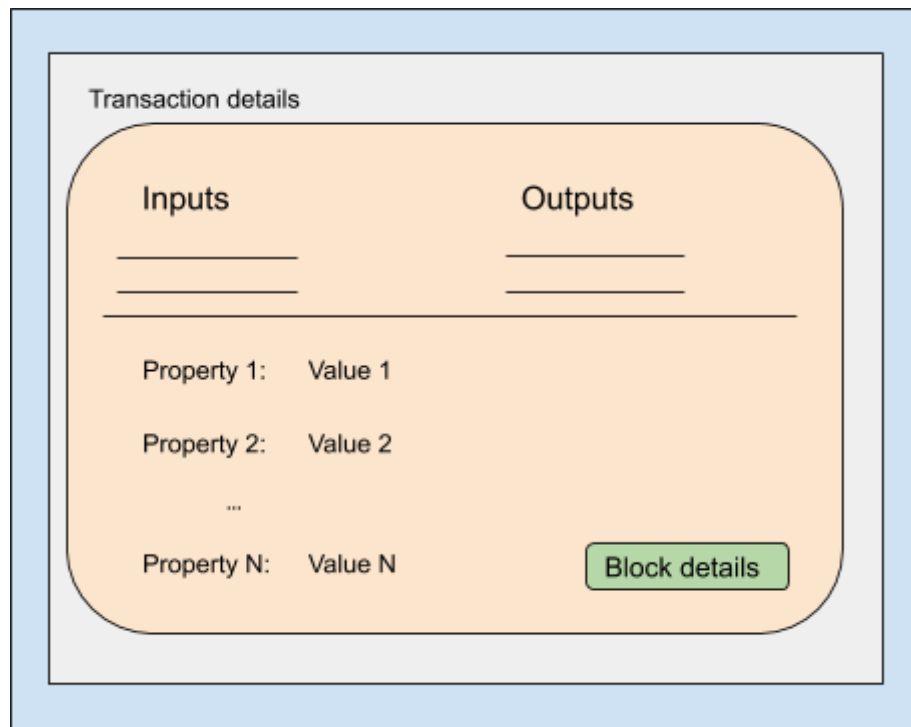


Image 39 - Transaction details new layout

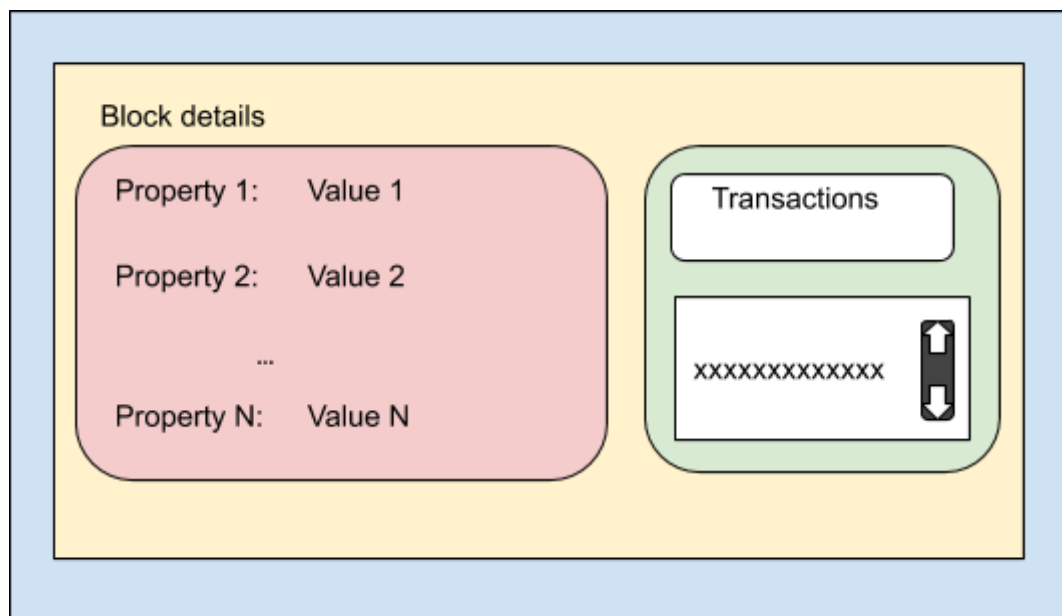


Image 40 - Block details new layout

Graph details

Main purpose of the redesigning process was based on this page. The reason for that was based on the previous layout, which was impractical for user experience. The element positioning did not contemplate relationships between nodes when participating in a

transaction and, moreover, there were no data associated. So, in order to transform this new design into a clear summary of the graphical representation displayed in the “Graph” tab, it was necessary the addition of compact and clear regions where users can distinguish between data associated to clusters, nodes, edges, as well as the usage of buttons allowing them to get data in real-time, such as address details, tx details and heuristic values.

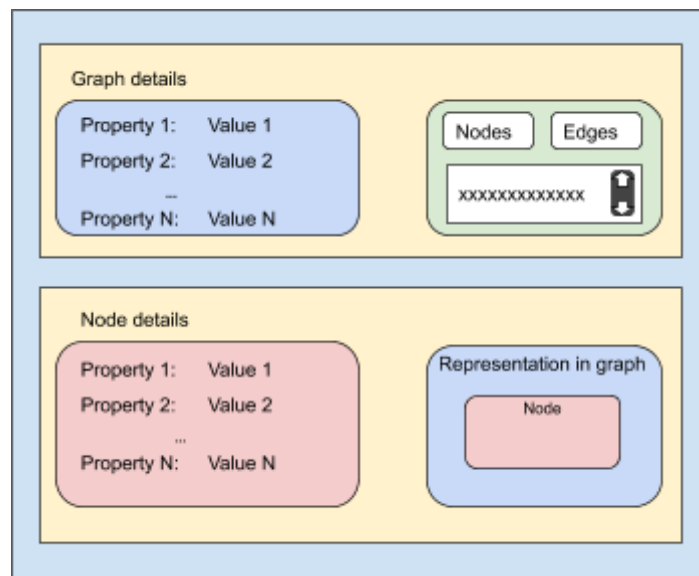


Figure - Graph details new layout

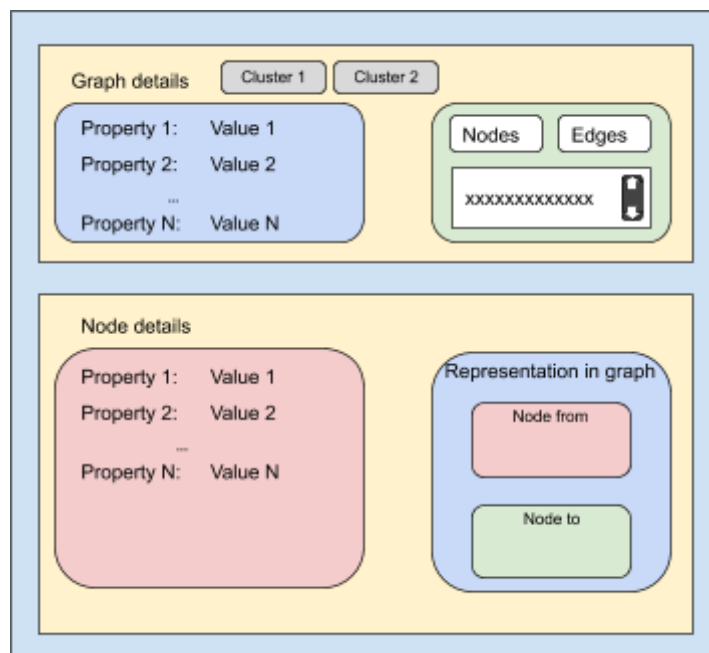


Image 41 - Cluster details new layout

Graph visualization

One of the main and distinctive traits Bitsights has to offer is the possibility of getting a visualized footprint of interactions between addresses and transactions that are not directly connected, which are listed and explained in three categories. As far as it concerns, building a graph with these specifications is a matter of displaying nodes and edges as previously explained. But, granting users the chance to interact with it is taking it to the next level: a graph is not only a simple image with innocuous content, it should be a tool that reinforces its interactions.

For that reason, many points have been taken into account: users have to easily choose whether to access elements details, watch the whole network of nodes and edges, or just download content to use later. Everything in one place, allowing an uncomplicated navigation through info. In order to fulfill that, the implementation was based on a tab navigator, clearly splitted in three sections: **Graph**, **Details** and **Download**.

As its name describes it, **Graph** section shows the visual layout of the graph, which differs depending on if it's an Address Distance, Related Address or a Relationship Cluster graph. Both of them were implemented relying on **NGX-Graph**, a graph visualization library for Angular.

Main challenge of this section was to prove that displaying big data in the form of a graph is possible without lacking user experience. For that reason, it was necessary to include some visual resources such as: a) usage of tooltips when hovering nodes and edges so as not to just present a plain graph with rectangles and arrows, b) implementation of a sidebar, which offers features like graph layout customization, zoom in/out, center graph or allow panning inside graph layout.

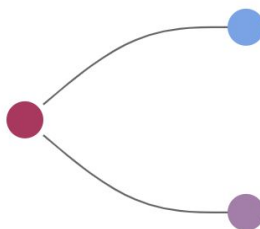


Image 42 - minigraph example representation obtained by using ngx-graph library.

Download graph

Although users have access to graphs and its details, it is also an interesting feature that offers them the possibility of downloading it. The available content for download that Bitsights provides is a .png file of the displayed graph using the DOT language. To fulfill this requisite, a graphviz library for angular was used: **d3-graphviz**. It allows rendering of SVG graphs from DOT sources, taking advantage of the graphviz package. And to save svg into a .png file another package was injected: **save-svg-as-png**.

Heuristics

To simplify code and design, this page has a similar approach as Search and Graph page. It has a search bar and a section with suggested addresses in order to search a specific piece of data previously tested: time-series balance, cluster balance, wallet probability and volume balance. Searching flow is practically the same as searching addresses, blocks, transactions or graphs with a slight difference: results are shown on the same page, so as not to force users to get dizzy about it. Users may take either one specific heuristic value or get them all at once. For this last case, users have not to be worried about waiting too much for all big responses, as requests are made in an asynchronous way, which leads in showing values as soon as responses are held.