UNIVERSITÀ DI PISA

Analysis of Bitcoin transactions

Matteo Pinna

Introduction

Transaction abstraction

The considered data-set contains transactions that are abstracted w.r.t. original Bitcoin transactions. All the housekeeping fields (e.g. *version*, *vin_sz*, *vout_sz*, size etc.) and some others (e.g. *lock_time*) have been removed. A real Bitcoin transaction would also have had a list of inputs and outputs in its metadata, in our case this information is implicit in the number of entries that can be found in the *inputs.csv* and *output.csv* files for a specific transaction id.
Moreover, in order to keep the data-set relatively small, some fields are a simplified/abstracted version of the original. Therefore, the hash representing a transaction id, input id and output id, as well as *scriptPubKey* and *signatureSig*, have been replaced with numeric identifiers.

Implementation

Validation and analysis phases have been done through programs developed in Python. External libraries that have been used are the following: *pandas* and *numpy* to convert the data-set into a *pandas.Dataframe* and to process it, both during validation and analysis, and *matplotlib* and *matplotlib.pyplot* in order to plot the results.

Validation

During the validation of the data-set several invalid transactions have been identified, for a total of 16:

- **Double spending:** at least one input of the transaction tried to use an output not present in the UTXO, hence a non-unspent output.
  - *Total: 9* (in one case, the *out_id* of the input did not exist at all)
- **Negative PK:** at least one output of the transaction had a negative $PK\_ID \neq -1$, hence it wasn't just the case of a non-standard script being used.
  - *Total: 1*
- **Invalid SIG-PK:** at least one *SIG_ID* of a specific input did not match the *PK_ID* of the output it tried to spend, hence it couldn't unlock the associated value. Transactions where a non-standard script was used have also been included.
  - *Total: 2*
- **Negative output:** at least one destination output had a negative *value*.
  - *Total: 1*
- **Inputs values $<$ Outputs values:** the total value associated to the inputs of the transactions was smaller than the total value being sent to the outputs.
  - *Total: 1*
- **Invalid Coinbase:** the total value associated to a Coinbase transaction was less than $50BTC$.
  - *Total: 2* (value was $25BTC$)

Notice that Coinbase transactions with $< 50BTC$ total value have been considered invalid due to the fact that the data-set starts from the genesis block and ends at the block mined on 29-12-2010 (block height $100,001$). Meanwhile, the first halving of Bitcoin happened around 2012 (block height $210,000$).

Analysis

UTXO

As of the last block:

- **Total number of UTXO:** $71,896$
- **Total UTXO value:** $5,000,800$ *(BTC)*

The UTXO with the highest associated value ($90,000 BTC$) is represented by the following 4-tuple:


```
"tx_id": 140,479,
"block_id": 90,532,
"output_index": 170,430,
"output_address": 138,895
```


Block occupancy

The block occupancy is defined as the number of transactions in each block in the entire period of time. In Figure 1 the block occupancy for each block, in Figure 2 the block occupancy distribution, and in Figure 3 the block occupancy over time considering a time step of 1 month.

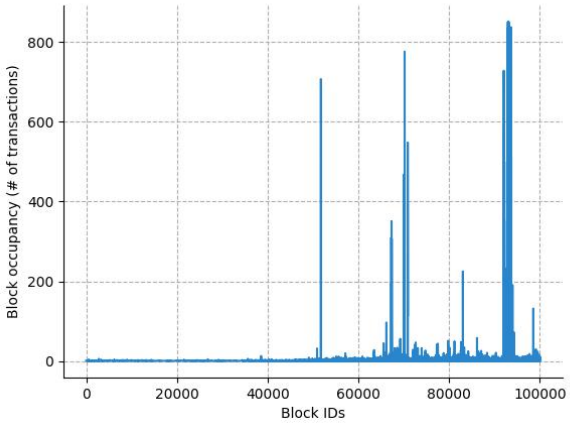

Figure 1: Block occupancy for each block

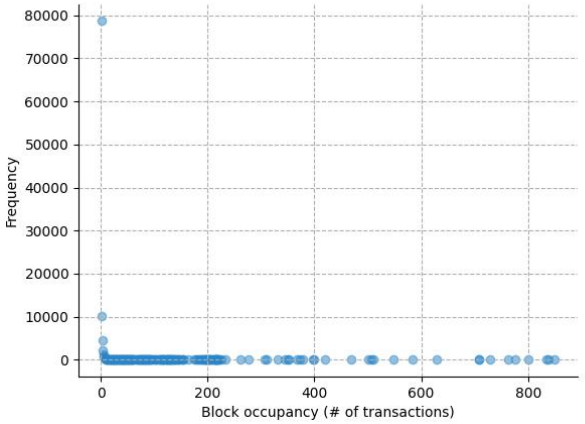

Figure 2: Block occupancy distribution

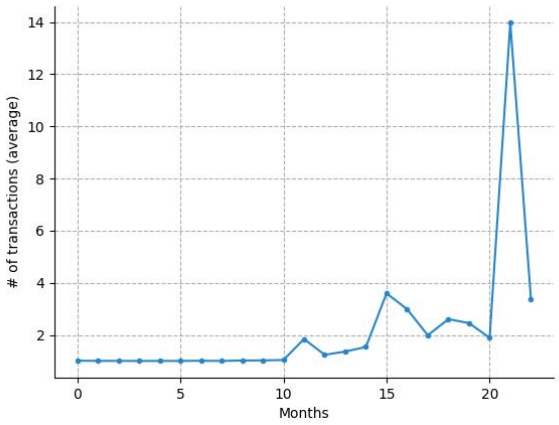

Figure 3: Block size (occupancy) over time

Bitcoin received

In Figure 4 the total amount of bitcoin received from each public key (address) that received at least one Coinbase transaction, in Figure 5 the distribution of such results.

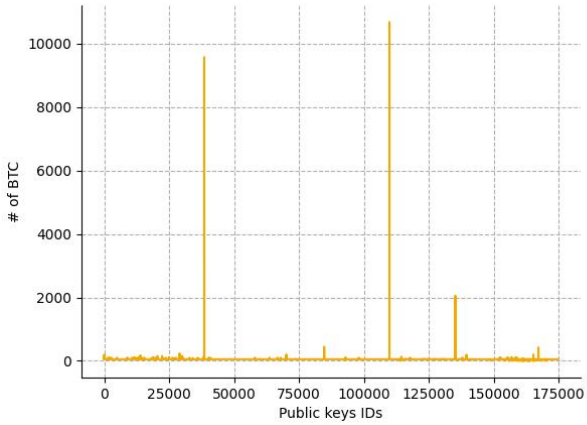

Figure 4: Received bitcoin for each public key

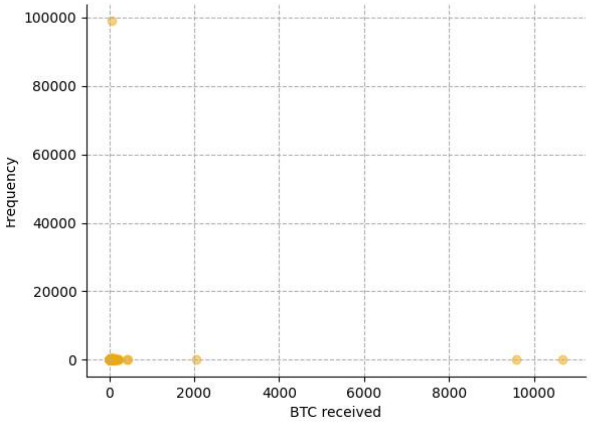

Figure 5: Received bitcoin distribution

Fees

In Figure 6 the fees for each transaction, in Figure 7 the fees distribution.

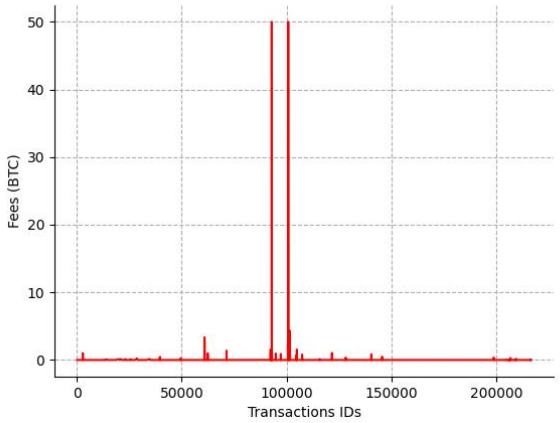

Figure 6: Fees for each transaction

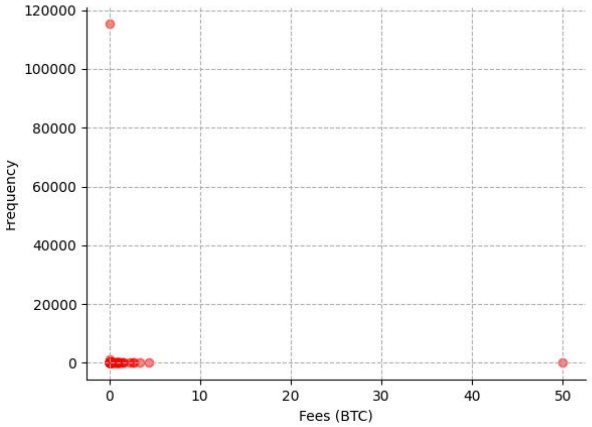

Figure 7: Fees distribution

Analysis of choice

As additional analysis, it has been decided to further analyze the relation between the fees of each transaction and some other aspects.

In Figure 8 the correlation between fees and total transaction value.
In Figure 9 the correlation between fees and total number of inputs, where the number of inputs of a transaction has been considered as an estimate of transaction complexity, since the size (Mb) was not available in our data-set.

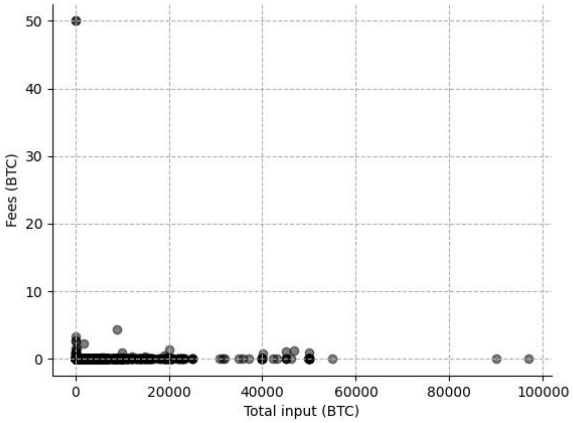

Figure 8: Mapping between fee and total value being transferred.

The results show that there isn't a correlation between the two parameters taken into account, hence the total transaction value does not warrant higher fees.

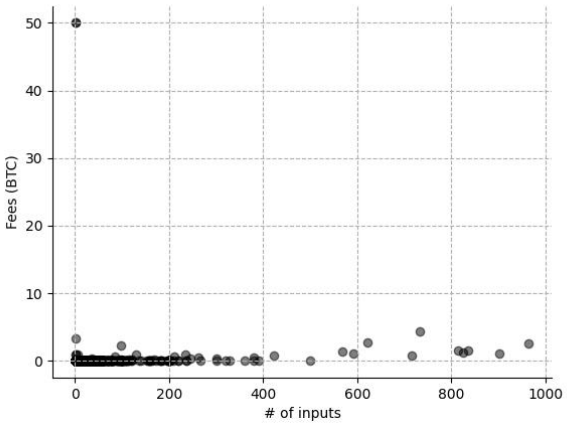

Figure 9: Mapping between fee and number of inputs.

In this case, it is possible to notice a stronger correlation, hence the complexity of a transaction impacts in some way the fees.

UNIVERSITÀ DI PISA

Kademlia DHT

Matteo Pinna

Introduction

Assuming a Kademlia network with ID size $n = 8$ bits and bucket size $k = 4$, the k-buckets of the peer with ID 11001010 are as follows:

- *k-bucket 7:* 01001111, 00110011, 01010101, 00000010
- *k-bucket 6:* 10110011, 10111000, 10001000
- *k-bucket 5:* 11101010, 11101110, 11100011, 11110000
- *k-bucket 4:* 11010011, 11010110
- *k-bucket 3:* 11000111
- *k-bucket 2:*
- *k-bucket 1:*
- *k-bucket 0:*

Notice that a bucket's list can contain at most k elements, in this case at most 4.

Solutions

Buckets and waiting lists

Messages from the following nodes arrive in this given order: 01101001, 10111000, 11110001, 10101010, 11100011, 11111111.
Assuming that *all nodes will always respond if pinged*:

01101001 (k-bucket 7):

- bucket is full, ping least recently contacted node in the list (first position)
- 01001111 answers back, hence it is pushed to the tail of the list
- new node 01101001 inserted in waiting list 7

10111000 (k-bucket 6):

- already present (not a new node), moved to the tail of the list

11110001 (k-bucket 5):

- bucket is full, ping least recently contacted node in the list (first position)
- 11101010 answers back, hence it is pushed to the tail of the list
- new node 11110001 inserted in waiting list 5

10101010 (k-bucket 6):

- bucket is not full, added to the tail of the list

11100011 (k-bucket 5):

- already present (not a new node), moved to the tail of the list

11111111 (k-bucket 5):

- bucket is full, ping least recently contacted node in the list (first position)
- 11101110 answers back, hence it is pushed to the tail of the list
- new node 11111111 inserted in waiting list 5

Notice that as of the last message, 11101110 is pinged because it is the current least recently contacted node. Since 11101010 was previously moved to the tail of the list when it answered the ping request.

In Table 1 the buckets, the orderings and the corresponding waiting lists after all messages have arrived.

Node becomes unreachable

The considered peer 11001010 detects that the peer 11101110 (k-bucket 5) cannot be reached anymore. Hence, 11101110 is deleted from k-bucket 5 and, since waiting list 5 is not empty, 11110001, which is at the first position of waiting list 5, is added to the tail of the list of k-bucket 5.

Bucket	List	Waiting List
k-bucket 7	00110011, 01010101, 00000010, 01001111	01101001
k-bucket 6	10110011, 10001000, 10111000, 10101010	
k-bucket 5	11110000, 11101010, 11100011, 11101110	11110001, 11111111
k-bucket 4	11010011, 11010110	
k-bucket 3	11000111	
k-bucket 2		
k-bucket 1		
k-bucket 0		

Table 1: Buckets, orderings and waiting lists as of the last message.

Lookup

In the event that a lookup for ID 11010010 occurs, the k-closest nodes must be picked. The ID 11010010 falls in k-bucket 4, however such bucket is not full since only two nodes are present: 11010011 and 11010110. Therefore, list 4 has to be extended with two more nodes in order to reach k entries, this is done by looking at close buckets: the other two closest nodes for the searched ID are: 11000111 (k-bucket 3) and 11110000 (k-bucket 5).
Then, a tuple (*IP address*, *UDP port*, *Node ID*) is returned for each of the k-closest nodes found.