UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology
Department of Computer Science

Master Thesis title

subtitle

—

Enrico Tedeschi

Master Thesis in Computer Science

Abstract

Contents

List of Figures

My list of definitions

/1

Introduction

2

Related Works

This chapter summarizes the most relevant papers or works that talks about Bitcoin, decentralized cryptocurrencies, concepts behind them and statistical analysis on the blockchain. In a previous paper that we wrote [41], we enhanced the importance of paying for having a certain bandwidth in the Bitcoin network. A paper from Peter R. Rizun [37], explains how a rational Bitcoin miner should select transactions from his node mempool, when creating a new block, in order to maximize his profit. Analysis on the blockchain in matter of scalability is showed in the Position Paper of Kyle Croman [22], they analyze how fundamental bottlenecks in Bitcoin limit the ability of its current peer-to-peer overlay network to support substantially higher throughputs and lower latencies. We are going to test the throughput as well, comparing it with the one showed in this paper.

2.1 A Transaction Fee Market Exists Without a Block Size Limit

This paper shows how a Bitcoin miner should select transactions from his node's mempool, when creating a new block, in order to maximize his profit in the absence of a block size limit. *Block space supply curve* and *mempool demand curve* are explained, and the paper shows how the supply and demand curves from classical economics are related to the derivatives of these two curves.

They claim that the block-size limit determines the transaction throughput and one of their concern regards whether, in the absence of such a limit or if that limit is far above the transactional demand, a *healthy transaction fee market* would develop which charges users the full cost to post transactions. The object of this paper is indeed to consider whether or not such a fee market is likely to emerge if miners, rather than the protocol, limit the block size. In this paper they derive the *miner's profit equation* and then they introduce two novel concepts called the *mempool demand curve* and the *block space subbly curve*.

2.1.1 Miner's Profit Equation

Every time a block is mined, the miner expects to generate a revenue $\langle V \rangle$ at hashing cost $\langle C \rangle$ to earn profit per block

$$\langle \Pi \rangle = \langle V \rangle - \langle C \rangle. \tag{2.1}$$

Miner's profit equation in 2.1 shows the gain of a miner $\langle \Pi \rangle$, where the hashing cost is represented as follows:

$$\langle C \rangle = \eta h T. \tag{2.2}$$

So the hashing cost $\langle C \rangle$ is directly dependent from the miner's individual hash rate, h, the cost per hash, η, and the creation time, T. Moreover, is important to consider the expectation value of a miner's revenue per block, this value is represented with $\langle V \rangle$ and is equal to the amount he would earn if he won the block multiplied by his probability of winning. So the expected revenue would be: $\langle V \rangle = (R + M)h/H$, where the amount he would earn is the sum of the block reward, R, and the transaction fees, M. His probability of winning, assuming all blocks propagating instantly, is equal to the ration of his hash rate, h, to the total hash rate of the Bitcoin network, H. The problem with this equation is that it does not reflect the miner's diminished chances of winning if he chooses to publish a block that propagates slowly to the other miners. If a miner finds first a valid block, but his solution is received after most miners are working on another, then his block will likely be discarded. This effect is called *orphaning*. The equation, considering the orphaning factor, $\mathbb{P}_{orphan}$, is the following:

$$\langle V \rangle = (R + M)\frac{h}{H}\left(1 - \mathbb{P}_{orphan}\right). \tag{2.3}$$

Where the chance that a block gets orphaned increases with the amount of time it takes the block to propagate to the other miners. Indeed, if τ is the block propagation time, the probability of orphaning is defined as:

$$\mathbb{P}_{orphan} = 1 - e^{-\frac{\tau}{T}}. \tag{2.4}$$

In conclusion the *miner's profit equation* is defined as:

$$\langle\Pi\rangle = (R + M)\frac{h}{H}e^{-\frac{\tau}{T}} - \eta h T \tag{2.5}$$

A *rational miner* selects which transactions to include in his block in a manner that maximizes the expectation value of his profit. This selection is explained with the *mempool demand curve* and the *block space supply curve*.

2.1.2 The Mempool Demand Curve

The set of transactions that still need to be approved and included in a block is called *mempool*. The mempool set is denoted with $\mathcal{N}$ and the number of transactions contained within it as n. According to the size limit, a block can select a $b \leq n$ transactions from $\mathcal{N}$ to create a new block $\mathcal{B} \subset \mathcal{N}$.

A block first includes transactions with an higher *fee density*. This last, is a ratio between the *transaction fee* and the *transaction size*. To construct the mempool demand curve, is necessary first sorting the mempool from greatest fee density to least and then associating an index $\{i : 1, 2, \ldots, n-1, n\}$ with each transaction in the resulting list.

The mempool demand curve will be then a graphical representation of the sum of the fees offered by each transaction in this sorted list:

$$M_{demand}(b) \equiv \sum_{i=1}^{b} fee_i, \tag{2.6}$$

and the sum of each transaction's size in bytes:

$$Q(b) \equiv \sum_{i=1}^{b} size_i. \tag{2.7}$$

2.1.3 The Block Space Supply Curve

The size of the block a miner elects to produce controls the fees he attempts to claim, $M(Q)$, and the propagation time he chooses to risk, $\tau(Q)$. The block space supply curve represents the fees a miner requires to cover the additional cost of supplying block space Q. This cost grows exponentially with the propagation time. The equation which represents this curve is the following:

$$M_{supply}(Q) = R\left(e^{\frac{\Delta\tau(Q)}{T}} - 1\right), \tag{2.8}$$

where $\Delta\tau(Q) \equiv \tau(Q) - \tau(0)$. The propagation time ,$\tau$, is just an esteem from the propagation delay versus the block size.

2.1.4 Maximizing the Miner's Profit

To maximize his profit, the miner construct a mempool demand curve and a space supply curve. The block size Q^* where the miner's surplus,

$$M_{demand} - M_{supply},$$

is largest represents the point of maximum profit. In the Figure 2.1 are represented the mempool demand curve and the space supply curve, calculated using our analitic system.

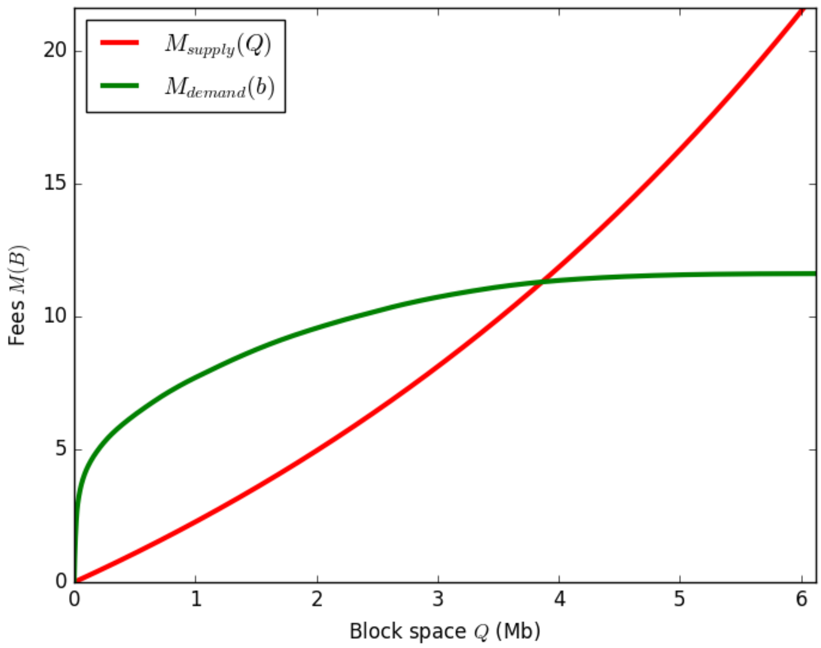

Figure 2.1: Maximizing the profit of a miner evaluating the unconfirmed transactions from the Bitcoin blockchain. Mempool demand, $M_{demand}(b)$, and space supply, $M_{demand}(Q)$, curve are represented with our analytics system, and the block space Q^* for a maximum profit is around 1 Mb. There the gap between the two curves is bigger.

3

Technical Background

4

Blockchain Analytics System

4.1 Blockchain Data Sources

4.2 System Architecture

4.2.1 Data Retrieval

4.2.2 Data Manipulations

4.2.3 Methods

4.3 Version Control

5

Blockchain Observations

6

Conclusions

6.1 Discussion

6.2 Future Implementation

6.3 Comments

show bibliography [36], [43], [19], [25], [34], [23], [10], [13], [35], [38], [29], [31], [15], [27], [42], [33], [1], [6], [21], [7], [40], [20], [39], [4], [8], [9], [14], [17], [16], [32], [18], [11], [22], [26], [28], [12], [2], [3], [37] [5], [41], [24], [30].

References

[1] Bitcoin api's, api-v1-client-python. https://github.com/blockchain/api-v1-client-python.

[2] Bitcoin mining hashing rate. https://blockchain.info/charts/hash-rate.

[3] Bitcoin nodes. https://bitnodes.21.co/.

[4] Bitocoin's blockchain website. https://blockchain.info.

[5] Construct a linear, no-fork, best version of the bitcoin blockchain. https://github.com/bitcoin/bitcoin/tree/master/contrib/linearize.

[6] Ethereum api's, pyethereum. https://github.com/ethereum/pyethereum.

[7] Ethereum wiki/patricia tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[8] Ethereum's blockchain analysis website. https://etherscan.io.

[9] Ethereum's blockchain website. https://etherchain.org/.

[10] Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper.

[11] Matplotlib for data plotting. https://matplotlib.org.

[12] Mining hardware comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[13] Ethereum foundation - the solidity contract-oriented programming language. Technical report, https://solidity.readthedocs.io/en/develop/, 2014.

[14] Bitcoin mining process. `http://bitcoinminer.com/`, 2015.

[15] Bitcoin website – mining. `https://www.bitcoinmining.com`, 2016.

[16] Ethereum project – website. `https://www.ethereum.org`, 2016.

[17] tradeblock.com – analysis on the blockchain. `https://tradeblock.com`, 2016.

[18] Alfred V. Aho and Jeffrey D. Ullman. *Foundations of Computer Science*. Computer Science Press, Inc., New York, NY, USA, 1992.

[19] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[20] Paul Baran. *On Distributed Communications: Introduction to Distributed Communication Networks*. The Rand Corporation, 1964.

[21] Georg Becker. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*. 2008.

[22] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. *On Scaling Decentralized Blockchains*, pages 106–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[23] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. *Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab*, pages 79–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[24] Jacob Donnelly. Bitcoin network still backlogged with tens of thousands of unconfirmed transactions, causing delays. *Bitcoin Magazine*, July 2015.

[25] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[26] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-ng: A scalable blockchain protocol. *CoRR*, abs/1510.02037, 2015.

[27] Rui Garcia, Rodrigo Rodrigues, and Nuno Preguiça. Efficient middleware for byzantine fault tolerant database replication. In *Proceedings of the*

Sixth Conference on Computer Systems, EuroSys '11, pages 107–122, New York, NY, USA, 2011. ACM.

[28] Begnaud Francis Hildebrand. *Introduction to Numerical Analysis: 2Nd Edition*. Dover Publications, Inc., New York, NY, USA, 1987.

[29] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[30] Nicolas Houy. The economics of bitcoin transaction fees. Working Papers 1407, Groupe d'Analyse et de Théorie Economique (GATE), Centre national de la recherche scientifique (CNRS), Université Lyon 2, Ecole Normale Supérieure, 2014.

[31] Håvard D Johansen, Robbert van Renesse, Ymir Vigfusson, and Dag Johansen. Fireflies: A secure and scalable membership and gossip service. *ACM Transactions on Computer Systems (TOCS)*, 33(2):5:1–5:32, May 2015.

[32] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[33] Aldelir Fernando Luiz, Lau Cheuk Lung, and Miguel Correia. Mitra: Byzantine fault-tolerant middleware for transaction processing on replicated databases. *SIGMOD Rec.*, 43(1):32–38, May 2014.

[34] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 254–269, New York, NY, USA, 2016. ACM.

[35] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 706–719, New York, NY, USA, 2015. ACM.

[36] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2008.

[37] Peter R. Rizun. A transaction fee market exists without a block size limit. Technical report, 2015.

[38] Chaitya B. Shah and Drashti R. Panchal. Secured hash algorithm-1: Review

paper. Technical report, Indus Institute of Technology and Engineering, Gujarat Technological University, 2014.

[39] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002.

[40] M. Swan. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, 2015.

[41] Enrico Tedeschi. Paying for bandwidth in blockchain internet applications. Technical report, UiT Arctic University of Norway, 2016.

[42] Ben Vandiver, Hari Balakrishnan, Barbara Liskov, and Samuel Madden. Tolerating Byzantine Faults in Transaction Processing Systems Using Commit Barrier Scheduling. In *ACM SOSP*, Stevenson, WA, October 2007.

[43] Dr. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, 2014.

A

Terminology

RLP: Stands for recursive length prefix. It is a serialization method for encoding arbitrary structured binary data (byte arrays).

KEC-256: Another serialization method generating a 256-bit hash.

full node: A full node in a decentralized digital currency peer-2-peer network, is a node that stores and processes the entirety of every block, storing locally the entire size of the blockchain.

light node: A light node in a decentralized digital currency peer-2-peer network, is a node that only stores the part of the blockchain it needs.

satoshi: Unit of the Bitcoin currency. 100,000,000 satoshi are 1 BTC (Bitcoin).

/B

Listing

Listing B.1: Smart contract transaction code in Ethereum.

```
function transfer(address _to, uint256 _value) {
        /* Add and subtract new balances */
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        }
```

Listing B.2: Example of a smart contract that rewards users who solve a computational puzzle [?].

```
contract Puzzle {
  address public owner ;
  bool public locked ;
  uint public reward ;
  bytes32 public diff ;
  bytes public solution ;

  function Puzzle () // constructor {
    owner = msg. sender ;
    reward = msg . value ;
    locked = false ;
    diff = bytes32 (11111); // pre - defined
       difficulty
    }

  function (){ // main code , runs at every
     invocation
    if ( msg. sender == owner ){ // update reward
    if ( locked )
    throw ;
    owner . send ( reward );
    reward = msg . value ;
    }
    else
      if ( msg . data . length > 0){ // submit a
         solution
      if ( locked ) throw ;
      if ( sha256 (msg. data ) < diff ){
          msg. sender . send ( reward ); // send
             reward
          solution = msg. data ;
          locked
          }}}}
```

Listing B.3: Application usage.

```
Usage: observ.py -t number
  -h | --help       : usage
  -i                : gives info of the blockchain in
    the file .txt
  -t number         : add on top a number of blocks.
    The blocks retreived will be the most recent
    ones. If the blockchain growth more than the
    block requested do -u (update)
  -e number         : append blocks at the end of the .
    txt file. Fetch older blocks starting from the
    last retrieved
  -P                : plot all
  -p start [end] : plot data in .txt file in a
    certain period of time, from start to end. If
    only start then consider from start to the end
    of the .txt file
  -R                : plot the regression and the
    models that predict the blockchain
  -r start [end] : plot the regression and the
    models in a certain period of time, from start
    to end. If only start then consider from start
    to the end of the .txt file
  -u                : update the local blockchain to
    the last block created
```

Listing B.4: Block object represented in Python according to api-v1-client-python to retrieve data on the blockchain. The function get_block() will return an object of this type [1].

```
class Block:
  def __init__(self, b):
    self.hash = b['hash']
    self.version = b['ver']
    self.previous_block = b['prev_block']
    self.merkle_root = b['mrkl_root']
    self.time = b['time']
    self.bits = b['bits']
    self.fee = b['fee']
    self.nonce = b['nonce']
    self.n_tx = b['n_tx']
    self.size = b['size']
    self.block_index = b['block_index']
    self.main_chain = b['main_chain']
    self.height = b['height']
    self.received_time = b.get('received_time', b['time'])
    self.relayed_by = b.get('relayed_by')
    self.transactions = [Transaction(t) for t in b['tx']]
    for tx in self.transactions:
        tx.block_height = self.height
```

Listing B.5: Transaction object represented in Python according to api-v1-client-python to retrieve data on the blockchain. The function get_transaction() will return an object of this type [1].

```python
class Transaction:
  def __init__(self, t):
    self.double_spend = t.get('double_spend', False)
    self.block_height = t.get('block_height')
    self.time = t['time']
    self.relayed_by = t['relayed_by']
    self.hash = t['hash']
    self.tx_index = t['tx_index']
    self.version = t['ver']
    self.size = t['size']
    self.inputs = [Input(i) for i in t['inputs']]
    self.outputs = [Output(o) for o in t['out']]

    if self.block_height is None:
      self.block_height = -1
```

Listing B.6: Json object returned from the method *get_block()* in the Bitcoin application programming interface (API) class blockexplorer.py [1]

```
hash : str
version : int
previous_block : str
merkle_root : str
time : int
bits : int
fee : int
nonce int
n_tx : int
size : int
block_index : int
main_chain : bool
height : int
received_time : int
relayed_by : string
transactions : array of Transaction objects
```

Listing B.7: Structure of the latest block retrieved. The function get_latest_block() will return an object with this structure.

```
class LatestBlock:
def __init__(self, b):
self.hash = b['hash']
self.time = b['time']
self.block_index = b['block_index']
self.height = b['height']
self.tx_indexes = [i for i in b['txIndexes']]
```

Listing B.8: Collecting data starting from the last element in the blockchain.txt file.

```
earliest_hash = get_earliest_hash()
get_blockchain(n, earliest_hash)

def get_blockchain(n, hash = None):
  [...]
  if (hash):  # start the retrieval from the hash
    append_end = True  # in that way the
       write_blockchain method knows that has to
       append blocks and not write them at the
       beginning
    last_block = blockexplorer.get_block(hash)
  [...]

def get_earliest_hash():
  hash_list = get_list_from_file("hash") # method to
      collect data from blockchain.txt file having
     as attribute "hash"
  length = len(hash_list)
  earliest_hash = hash_list[length - 1]
  return earliest_hash
```

Listing B.9: Calling `blockchain.info` through python API and retrieving part of the blockchain.

```
from blockchain import blockexplorer
# get the last block
last_block = blockexplorer.get_latest_block()
hash_last_block = last_block.hash

# current block now is the last block
current_block = blockexplorer.get_block(
    hash_last_block)
```

Listing B.10: How read bandwidth is calculated, using the function *datetime.now*() before and after the API call.

```
start_time = datetime.datetime.now() # ---------
current_block = blockexplorer.get_block(
    current_block.previous_block)
end_time = datetime.datetime.now() # ---------
time_to_fetch = end_time - start_time
time_in_seconds = get_time_in_seconds(time_to_fetch)

#latency
fetch_time_list.append(time_in_seconds)

# calculate Bandwidth with MB/s
block_size = float(current_block.size) / 1000000
bandwidth = block_size / time_in_seconds
bandwidth_list.append(bandwidth)
```

Listing B.11: Function that get the average write bandwidth of the block, calculating the time for each transaction to be visible in the public ledger of data.

```python
def get_avg_transaction_time(block):
  # take transactions the block
  transactions = block.transactions

  # get block time -- when it is visible in the blockchain, so when it was created
  block_time = block.time

  # list of the creation time for all the transaction in the block
  transactions_time_list = []

  # list of the time that each transaction needs before being visible in the blockchain
  time_to_be_visible = []

  for t in transactions:
    transactions_time_list.append(float(t.time))

  for t_time in transactions_time_list:
    time_to_be_visible.append(float(block_time - t_time))

  average_per_block = sum(time_to_be_visible) / len(time_to_be_visible)
  return average_per_block
```

Listing B.12: How the file.write is performed in the blockchain analytics system. Differences with add and append.

```
if(append):
  for i in range(n):
    file.write("block_informations")

else: # add on top
  hash_list_in_file = get_list_from_file("hash")
  first_hash = hash_list_in_file[0]
  elements = len(hash_list_in_file)
  last_hash = hash_list_in_file[elements-1]
  met_first = False
  with io.FileIO(file_name, "a+") as file:
    file.seek(0)  # place at the beginning of the file
    existing_lines = file.readlines()  # read the already existing lines
    file.seek(0)
    file.truncate()  # delete all the file
    file.seek(0)
    i = 0
    while (i < n):
      if (first_hash == hash[i]):
        met_first = True
      while((met_first == False) and (i < n)):
        # append on top
        file.write("block_informations")
        i = i + 1
        if ((i < n) and (first_hash == hash[i])):
          met_first = True
      # when the block retrieved meets the one alread in the blockchain write the old file
      file.writelines(existing_lines)
```

Listing B.13: Changing all the values inside a Python list in one code line.

```
# size_list from byte to MB
size_list[:] = [x / 1000000 for x in size_list]
# time_list from seconds in minutes
time_list[:] = [x / 60 for x in time_list]
```

Listing B.14: Method that allows, given an attribute present in the blockchain.txt file, to create a list containing informations only about this quality using *regular expressions*.

```
hash_list = get_list_from_file("hash")

def get_list_from_file(attribute):
  list_to_return = []
  if (os.path.isfile("blockchain.txt")):
    # open the file and read in it -- blockchain_file
    with open("blockchain.txt", "r") as blockchain_file:
      for line in blockchain_file:
      # regular expression that puts in a list the line just read: eg. ['hash', '<block_hash>']
        list = re.findall(r"[\w']+", line)
        # list[0] --> contains the attribute
        # list[1] --> contains the value
        if ((list) and (list[0] == attribute)):
          list_to_return.append(list[1])
  return list_to_return
```

Listing B.15: Generation of the growing size and time lists. Calculated following the Equations **??**, **??**.

```
def create_growing_time_list(time_list):
  reversed_time_list = time_list[::-1]
  time_to_append = 0
  previous_time = 0
  growing_time_list = []
  growing_time_list.append(previous_time)
  for time_el in reversed_time_list:
    time_to_append = (float(time_el) / (60 * 60)) +
       previous_time  # time in hours
    growing_time_list.append(time_to_append)
    previous_time = time_to_append
  return growing_time_list

def create_growing_size_list(size_list):
  reversed_size_list = size_list[::-1]
  growing_size_list = []
  value_to_append = 0
  size_back = 0
  growing_size_list.append(value_to_append)
  for size_el in reversed_size_list:
    value_to_append = size_el + size_back
    growing_size_list.append(value_to_append)
    size_back = value_to_append
  return growing_size_list
```

Listing B.16: Example of a polynomial interpolation of two lists using NumPy libraries.

```
import numpy as np

model = np.polyfit(list1, list2, deg) # polynomial
   interpolation between list1 and list2 with the
   degree of the return polynomial
# deg = 1 --> linear interpolation
# deg = 2 --> quadratic
# deg = 3 --> cubic
# ...
predicted = np.polyval(model, list1)
plt.plot(list1, predicted, 'b-', label="pol_interp")
```

Listing B.17: Check the status of the local blockhain, True if valid, False if not.

```
def check_blockchain():
  check = True
  list = get_list_from_file("height")
  number = int(list[0])
  length_list = len(list)
  for i in range(length_list):
    if (number != int(list[i])):
      check = False
    number = number - 1
  return check
```