

## Master Thesis title

*subtitle*

---

**Enrico Tedeschi**

*Master Thesis in Computer Science*





# **Abstract**



# Contents

<b>Abstract</b>	<b>i</b>
<b>My list of definitions</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Method / Context . . . . .	1
1.3 Outline . . . . .	1
<b>2 Related Works</b>	<b>3</b>
2.1 A Transaction Fee Market Exists Without a Block Size Limit . . . . .	3
<b>3 Technical Background</b>	<b>5</b>
<b>4 Blockchain Analytics System</b>	<b>7</b>
4.1 Blockchain Data Sources . . . . .	7
4.2 System Architecture . . . . .	7
4.2.1 Data Retrieval . . . . .	7
4.2.2 Data Manipulations . . . . .	7
4.2.3 Methods . . . . .	7
4.3 Version Control . . . . .	7
<b>5 Blockchain Observations</b>	<b>9</b>
5.1 Blockchain Growth . . . . .	9
5.2 Retrieval Block Time . . . . .	9
5.3 Block Analysis . . . . .	9
5.4 Bandwidth . . . . .	9
5.5 Block Fee . . . . .	9
5.6 Models . . . . .	9
<b>6 Conclusions</b>	<b>11</b>
6.1 Discussion . . . . .	11
6.2 Future Implementation . . . . .	11
6.3 Comments . . . . .	11

<b>References</b>	<b>13</b>
<b>A Terminology</b>	<b>17</b>
<b>B Listing</b>	<b>19</b>

# **My list of definitions**



# / 1

## Introduction

**1.1 Problem Statement**

**1.2 Method / Context**

**1.3 Outline**



# /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 [39], we enhanced the importance of paying for having a certain bandwidth in the Bitcoin network. A paper from Peter R. Rizun [35], 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.

### 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.



# /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**

- 5.1 Blockchain Growth**
- 5.2 Retrieval Block Time**
- 5.3 Block Analysis**
- 5.4 Bandwidth**
- 5.5 Block Fee**
- 5.6 Models**



# /6

## Conclusions

### 6.1 Discussion

### 6.2 Future Implementation

### 6.3 Comments

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



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

- [27] Begnaud Francis Hildebrand. *Introduction to Numerical Analysis: 2Nd Edition*. Dover Publications, Inc., New York, NY, USA, 1987.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system,” <http://bitcoin.org/bitcoin.pdf>, 2008.
- [35] Peter R. Rizun. A transaction fee market exists without a block size limit. Technical report, 2015.
- [36] 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.
- [37] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002.
- [38] M. Swan. *Blockchain: Blueprint for a New Economy*. O’Reilly Media, 2015.

- [39] Enrico Tedeschi. Paying for bandwidth in blockchain internet applications. Technical report, UiT Arctic University of Norway, 2016.
- [40] 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.
- [41] Dr. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical report, 2014.



## 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-to-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-to-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).





## List<sup>ing</sup>

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

```
1 function transfer(address _to, uint256 _value) {
2     /* Add and subtract new balances */
3     balanceOf[msg.sender] -= _value;
4     balanceOf[_to] += _value;
5 }
```

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

```

1 contract Puzzle {
2     address public owner ;
3     bool public locked ;
4     uint public reward ;
5     bytes32 public diff ;
6     bytes public solution ;
7
8     function Puzzle () // constructor {
9         owner = msg. sender ;
10        reward = msg . value ;
11        locked = false ;
12        diff = bytes32 (11111); // pre - defined
13            difficulty
14    }
15
16    function (){ // main code , runs at every
17        invocation
18        if ( msg. sender == owner ){ // update reward
19            if ( locked )
20                throw ;
21            owner . send ( reward );
22            reward = msg . value ;
23        }
24        else
25            if ( msg . data . length > 0){ // submit a
26                solution
27            if ( locked ) throw ;
28            if ( sha256 (msg. data ) < diff ){
29                msg. sender . send ( reward ); // send
                    reward
                solution = msg. data ;
                locked
            }}}}}
```

**Listing B.3:** Application usage.

```

1 Usage: observ.py -t number
2   -h | --help      : usage
3   -i              : gives info of the blockchain in
4     the file .txt
5   -t number       : add on top a number of blocks.
6     The blocks retrieved will be the most recent
7     ones. If the blockchain growth more than the
8     block requested do -u (update)
9   -e number       : append blocks at the end of the .
10    txt file. Fetch older blocks starting from the
11    last retrieved
12   -P              : plot all
13   -p start [end] : plot data in .txt file in a
14     certain period of time, from start to end. If
15     only start then consider from start to the end
16     of the .txt file
17   -R              : plot the regression and the
18     models that predict the blockchain
19   -r start [end] : plot the regression and the
20     models in a certain period of time, from start
21     to end. If only start then consider from start
22     to the end of the .txt file
23   -u              : update the local blockchain to
24     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].

```
1 class Block:
2     def __init__(self, b):
3         self.hash = b['hash']
4         self.version = b['ver']
5         self.previous_block = b['prev_block']
6         self.merkle_root = b['mrkl_root']
7         self.time = b['time']
8         self.bits = b['bits']
9         self.fee = b['fee']
10        self.nonce = b['nonce']
11        self.n_tx = b['n_tx']
12        self.size = b['size']
13        self.block_index = b['block_index']
14        self.main_chain = b['main_chain']
15        self.height = b['height']
16        self.received_time = b.get('received_time', b['time'])
17        self.relayed_by = b.get('relayed_by')
18        self.transactions = [Transaction(t) for t in b['tx']]
19        for tx in self.transactions:
20            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].

```

1  class Transaction:
2      def __init__(self, t):
3          self.double_spend = t.get('double_spend', False)
4          self.block_height = t.get('block_height')
5          self.time = t['time']
6          self.relayed_by = t[' relayed_by ']
7          self.hash = t['hash']
8          self.tx_index = t['tx_index']
9          self.version = t['ver']
10         self.size = t['size']
11         self.inputs = [Input(i) for i in t['inputs']]
12         self.outputs = [Output(o) for o in t['out']]
13
14         if self.block_height is None:
15             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.

```

1 class LatestBlock:
2     def __init__(self, b):
3         self.hash = b['hash']
4         self.time = b['time']
5         self.block_index = b['block_index']
6         self.height = b['height']
7         self.tx_indexes = [i for i in b['txIndexes']]
```

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

```

1 earliest_hash = get_earliest_hash()
2 get_blockchain(n, earliest_hash)
3
4 def get_blockchain(n, hash = None):
5     [...]
6     if (hash): # start the retrieval from the hash
7         append_end = True # in that way the
8             write_blockchain method knows that has to
9                 append blocks and not write them at the
10                  beginning
11     last_block = blockexplorer.get_block(hash)
12     [...]
13
14 def get_earliest_hash():
15     hash_list = get_list_from_file("hash") # method to
16         collect data from blockchain.txt file having
17             as attribute "hash"
18     length = len(hash_list)
19     earliest_hash = hash_list[length - 1]
20     return earliest_hash
```

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

```

1 from blockchain import blockexplorer
2 # get the last block
3 last_block = blockexplorer.get_latest_block()
4 hash_last_block = last_block.hash
5
6 # current block now is the last block
7 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.

```

1 start_time = datetime.datetime.now() # -----
2 current_block = blockexplorer.get_block(
    current_block.previous_block)
3 end_time = datetime.datetime.now() # -----
4 time_to_fetch = end_time - start_time
5 time_in_seconds = get_time_in_seconds(time_to_fetch)
6
7 #latency
8 fetch_time_list.append(time_in_seconds)
9
10 # calculate Bandwidth with MB/s
11 block_size = float(current_block.size) / 1000000
12 bandwidth = block_size / time_in_seconds
13 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.

```
1 def get_avg_transaction_time(block):
2     # take transactions the block
3     transactions = block.transactions
4
5     # get block time -- when it is visible in the
6     # blockchain, so when it was created
7     block_time = block.time
8
9     # list of the creation time for all the
10    # transaction in the block
11    transactions_time_list = []
12
13    # list of the time that each transaction needs
14    # before being visible in the blockchain
15    time_to_be_visible = []
16
17    for t in transactions:
18        transactions_time_list.append(float(t.time))
19
20        for t_time in transactions_time_list:
21            time_to_be_visible.append(float(block_time -
22                t_time))
23
24    average_per_block = sum(time_to_be_visible) / len(
25        time_to_be_visible)
26
27    return average_per_block
```

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

```

1 if (append):
2     for i in range(n):
3         file.write("block_informations")
4
5 else: # add on top
6     hash_list_in_file = get_list_from_file("hash")
7     first_hash = hash_list_in_file[0]
8     elements = len(hash_list_in_file)
9     last_hash = hash_list_in_file[elements - 1]
10    met_first = False
11    with io.FileIO(file_name, "a+") as file:
12        file.seek(0) # place at the beginning of the
13        file
14        existing_lines = file.readlines() # read the
15        already existing lines
16        file.seek(0)
17        file.truncate() # delete all the file
18        file.seek(0)
19        i = 0
20        while (i < n):
21            if (first_hash == hash[i]):
22                met_first = True
23            while((met_first == False) and (i < n)):
24                # append on top
25                file.write("block_informations")
26                i = i + 1
27                if ((i < n) and (first_hash == hash[i])):
28                    met_first = True
# when the block retrieved meets the one
# already 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.

```

1 # size_list from byte to MB
2 size_list[:] = [x / 1000000 for x in size_list]
3 # time_list from seconds in minutes
4 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*.

```
1 hash_list = get_list_from_file("hash")
2
3 def get_list_from_file(attribute):
4     list_to_return = []
5     if (os.path.isfile("blockchain.txt")):
6         # open the file and read in it --
7         blockchain_file
8         with open("blockchain.txt", "r") as
9             blockchain_file:
10                for line in blockchain_file:
11                    # regular expression that puts in a list the
12                    # line just read: eg. ['hash', '<block_hash
13                    >']
14                    list = re.findall(r"\w'+'\w", line)
15                    # list[0] --> contains the attribute
16                    # list[1] --> contains the value
17                    if ((list) and (list[0] == attribute)):
18                        list_to_return.append(list[1])
19
20 return list_to_return
```

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

```

1 def create_growing_time_list(time_list):
2     reversed_time_list = time_list[::-1]
3     time_to_append = 0
4     previous_time = 0
5     growing_time_list = []
6     growing_time_list.append(previous_time)
7     for time_el in reversed_time_list:
8         time_to_append = (float(time_el) / (60 * 60)) +
9             previous_time # time in hours
10        growing_time_list.append(time_to_append)
11        previous_time = time_to_append
12    return growing_time_list
13
14 def create_growing_size_list(size_list):
15     reversed_size_list = size_list[::-1]
16     growing_size_list = []
17     value_to_append = 0
18     size_back = 0
19     growing_size_list.append(value_to_append)
20     for size_el in reversed_size_list:
21         value_to_append = size_el + size_back
22         growing_size_list.append(value_to_append)
23         size_back = value_to_append
24     return growing_size_list

```

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

```

1 import numpy as np
2
3 model = np.polyfit(list1, list2, deg) # polynomial
4             interpolation between list1 and list2 with the
5             degree of the return polynomial
6 # deg = 1 --> linear interpolation
7 # deg = 2 --> quadratic
8 # deg = 3 --> cubic
9 # ...
8 predicted = np.polyval(model, list1)
9 plt.plot(list1, predicted, 'b-', label="pol_interp")

```

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

```
1 def check_blockchain():
2     check = True
3     list = get_list_from_file("height")
4     number = int(list[0])
5     length_list = len(list)
6     for i in range(length_list):
7         if (number != int(list[i])):
8             check = False
9             number = number - 1
10    return check
```



