Bitcoin Transaction: From the Creation to Validation, a Protocol Overview

Valentin Vallois
Beamap
Sopra Steria Group
Paris, France
vvallois@beamap.fr

Dr. Fouad Amine Guenane
Beamap
Sopra Steria Group
Paris, France
fguenane@beamap.fr

***Abstract*—In the last decade, we have seen the emergence of the Bitcoin crypto-currency, where the Blockchain technology builds trust transaction after transaction through a validation process. Academics focused their research on security flaws or improvement, but we identify a lack of literature on how the validation of transaction and block proceeds. In fact, understanding the validation will help the conception of new blockchain features and allow users to recognize all security risks. This paper aims at a deep comprehension and detailed analysis of the validation in the Bitcoin protocol.**

***Index Terms*— Bitcoin, Blockchain, Database, Transaction, Validation**

I. Introduction

FOR the past few years the blockchain technology has been the new trend in the computer world. Some call it the new Internet or even the new Linux [1]. Bitcoin, being the first implementation of blockchain technology, sets the standard for the industry. The goal of this paper is to produce a state of the art with the objective of simulating a blockchain in a manufacturing environment combined with IoT. We identify the following challenges that IoT is expected to resolve [2], [3].

- *Autonomous decision-making*. IoT devices don't have much computational capacity and they have to rely on external entities for decision-making. We can imagine an IoT device connected to a blockchain. While it is still an external entity, the blockchain stores smart contracts that will react to the information sent by IoT devices.
- *Standardization*. IoT come from different constructors and use different means of communications. They hence need a standard layer of communication, and blockchain can be that layer. Because a blockchain stores data, we can design an architecture built around it.
- *Safe, reliable and effective communication*. IoT are subjected to interferences, especially when using wireless communication. Blockchain systems are built for unreliable networking and trustless communications.

We believe that Bitcoin presents the right qualities for this kind of environment, but we will not talk about the proof of work, as it's inefficient for a private utilization and for the IoT [4], [5]. We judge that the data structure of the bitcoin is well designed and the validation process has proven is robustness through many years of existence. We chose to study Bitcoin. With instead of a chain-like structure we can use something like a tangle [6] that is more adapted for a fast and live environment. And use the Bitcoin inspired validation process for the verification of transactions. For all those reasons we choose to write this paper about the validation process.
First, we will focus on the basic mechanisms of the Bitcoin technology. Second we will explain the different data structures. At least we will show the interaction between them.

We present you with an analysis based on the bitcoin core v0.14 [7] as it's the most used software on the Bitcoin network. As we will focus on software we will not talk about the networking protocol. For further informations refer to [8].

II. Bitcoin

In this section, we will overview the basic concepts of the Bitcoin system. More emphasis on the transaction and its validation will occur later in this paper. In this study we use "Bitcoin" referring to the system or the technology and "bitcoin(BTC)" when referring to the token/currency associated with the technology. A bitcoin is divided in satoshi, 1 BTC = 100,000,000 satoshi.

Bitcoin is first of all a network of nodes, each one storing the blockchain. The Bitcoin uses a token (the "bitcoin") to entice users to maintain the network. New bitcoins are minted when a block is created. This incentive is really important to drive the system, as it pushes users to stay honest not to lose money [9], [10]. The information about ownership and how many tokens there are is stored in the blockchain. Everyone can verify the content of the blockchain and this content is replicated on each node of the network. As Bitcoin runs in a trustless environment, verification of information in transit is a key component for the well being of the system.

A. Transaction

A transaction is a data structure that stores the transfer of tokens between two parties. This data will live on the network until a miner adds it to a block.

Bitcoin was made for exchanging tokens with a monetary value [11], and the transaction is the vehicle of this exchange. We will present a more detailed explanation of the creation and verification process later in this paper.

978-1-5386-1332-0/2017/$31.00 ©2017 IEEE

B. Block

Updates on the database need to be in a batch form to ease synchronization between nodes. Bitcoin uses blocks, which are a data structure storing one or many transactions. Each block is created by a miner, and anyone can be a miner, connected to the Bitcoin network. The miner's goal is to find the solution to a crypto puzzle that verifies the creation of a block. A miner can either run the bitcoin core client or use a miner software. In both cases, the computer resources will be used for creating a valid block. The creator of such a block is rewarded with an amount of bitcoin. Thus exists a competition between the miners for creating a block as only one block will be append to the blockchain. The average network output is of one block every 10 minutes. Miners may have a different vision of the available transactions to add inside a block, because of the propagation of the information and the high liveliness of the network [12]. Before being added in a block, the transactions are stored in a pool (called mempool) of available transactions. This pool is saved in the RAM of the node, so if the node shuts down this information is lost. A new mined block is then sent on the network and will be broadcasted if considered valid. A transaction is called confirmed when it reaches a certain amount of maturity. Maturity is measured by the number of blocks between the one block where the transaction is located and the last block found. The Bitcoin documentation recommends a 6 blocks confirmation to guarantee the presence of the transaction in the blockchain.

C. Blockchain

Blocks are linked to each other through a field in their structure which contains a hash of the previous block: his parent. This allows for the stored information to appear in chronological order like on a ledger. Therefore the transactions have an order and keeping the total value of an account is done easily by looking up the blockchain and adding the results of those transactions.

The blockchain is organized in a tree form that spans from the root to the tip of the longest branch. The longest path to the origin is called the blockchain, all shorter branches are not considered legitimate and store information that might not be in the blockchain. We call the intersection of two branches a fork. The root is called the genesis block; it's the only block without a predecessor. There are four types of blocks. A "normal block" is part of the blockchain of which parent is also part of. "Invalid blocks" are rejected during the validation process and are then deleted. "Stale blocks" are not part of the longest chain, they are however valid and have a parent in the blockchain. They are always in the shortest branch of a fork. The last one is the "orphan block"; it is a valid block with an unknown parent. It mostly appears when the parent didn't have enough time to propagate through the network. When a miner creates a block, most of the time the parent is the tip of the blockchain. A miner can create a block even though the parent isn't yet correctly propagated through the network, which doesn't render the block invalid.

D. Blockchain forks

Forks are a common occurrence in a blockchain environment. A fork happens when two or more miners create a block at the same time and those two blocks have the same parent. The problem is that only the longest branch is the correct database. So at the time a fork occur the fate of the blockchain is decided by the next block. Whichever parent the miner chooses next depends on the blocks located in the miner's node. As the mining process continues the majority of the network will chose the longest chain. This selection depends mostly on the propagation of the information and the miners. Blocks present in the short branch are still saved in the blockchain database but aren't part of it. They are saved because miners can use them as parents for their blocks and with enough computational power they can take over the longest branch. Thus, unconfirming all the transactions inside of it that aren't inside the new branch. A transaction is never really confirmed but the probability increases exponentially after each new block is created [11]. This process is explained later in this paper.

III. DATA STRUCTURES

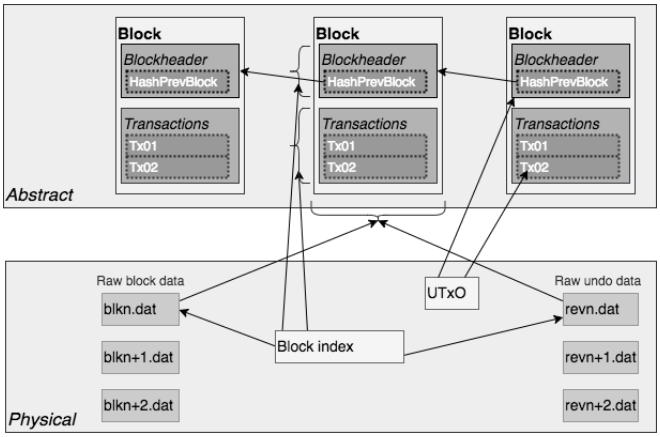

Fig. 1. Architecture and interaction between the data of Bitcoin Core

A. Transaction

Blockchain and Bitcoin are innovative from the traditional centralized bank system as the notion of account is absent. In bitcoin you have outputs or coins that are credited to a public address (linked to a pair of public key/private key). If you can prove that you own that public address, with a cryptographic signature, for example, you can spend the coin. It also means that users don't need to connect to each other simultaneously. It's a send and forgets system, as you publish the transaction on the network, it is added to the blockchain after an indeterminate time.

1) Transaction Data Structure

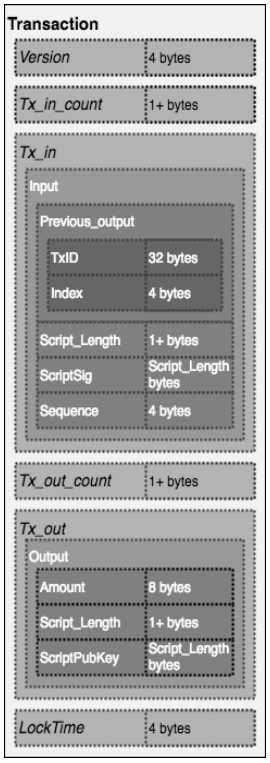

Fig. 2. Transaction data structure, in this example there is only one input and one output

A transaction is composed of six fields. The first one contains the version number, which indicates the rules followed by the transaction for validation. The second one contains the number of inputs inside the transaction which is followed by a list of inputs. Originating from previous outputs from other transactions. A coinbase transaction does not have input, but a field of 100 bytes where the extra nonce is stored and it can also store small message. The fourth field contains the number of outputs and the fifth field contains the list of outputs, at least one output is necessary. A coinbase transaction sends the block reward in an output targeting the miner. The last field contains the LockTime that sets a time limit until the outputs of the transaction can be spent. This limit can be in blocks or in standard UNIX time.

An input (Tx_in) is composed of 4 fields. The first one contains a reference to a previous output, and is called an outpoint. The second one contains information about the length of the ScriptSig with a size going from1 byte to a maximum of 10,000 bytes. The next field contains the ScriptSig, used to prove the ownership of the output. The last field contains the sequence number which is deprecated; it is used to verify the TimeLock with a default value of 0xFFFFFFFF.

A Previous_output or outpoint is composed of the TxID of a previous transaction and it is stored inside the UTxO. The TxID is the hash (SHA256) of the transaction. Next comes the index of the output inside the transaction, as there might be multiple outputs.

An output is composed of 3 fields. The first one contains the amount of bitcoin in satoshi of the output. The second one contains information about the length of the ScriptPubKey with a size going from1 byte to a maximum of 10,000 bytes. It's followed by the ScriptPubKey, which determines who can spend the output. One user can hold more than one private key, hence the destination of an output can be the sender himself or someone else, which means that the user can send bitcoin to themself. It can help increase the anonymity of the user. As it's hard to find who is the owner of a public key. For example, Alice sends 10 BTC to Bob and uses a previous output of 50 BTC. The transaction is composed of one output of 10 BTC to Bob and one of 40 BTC to Alice using a different public key than the input. Someone who looks into the transaction can't tell if Alice owns the 10 or 40 BTC output. It can only be true if Alice and Bob don't reuse the same public keys for receiving bitcoin. A good security practice is to get a new pair of keys for each new transaction.

2) Script

The Bitcoin protocol allows for simple computation to be added to a transaction. The script uses a restrictive language designed by the developer of the Bitcoin protocol. The language is stack based and designed to be stateless and not turing-complete.

The common use of the script for standard transaction is the Pay To Public Key Hash (P2PKH) and a more sophisticated script is the Pay To Script Hash (P2SH). Both of those scripts validate the proof of ownership when they are executed along a ScriptSig.

1) ScriptSig

This script can only push data on the stack, usually a signature and a public key. If the script executes any kind of functions, honest nodes will then discard the transaction.

2) P2PKH

This script is a verification of the hash of the public key and the signature of the owner of the output. The hash is RIPEMD-160(SHA256(PubKey)).

The script:

```
OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

The ScriptSig associated:

```
<sig> <pubkey>
```

3) P2SH

This script is a verification of the hash of a script named redeemScript and itself. The redeemScript is used for making multi signature verification. The redeemScript can store public keys and execute more complex verifications.

The script:

```
OP_HASH160 <Hash160(redeemScript)> OP_EQUAL
```

The ScriptSig associated:

```
<sig> [sig] [sig...] <redeemScript>
```

B. Block

Blocks are the main part of the blockchain which is the structure that stores the transactions. A block always refers to a parent, this relationship is the link of the chain.

1) Block data structure

A block is composed of three fields; the first one contains the block header, the second one contains the number of transactions inside the block, and the last one contains a list of the transactions. The number of transactions is limited by their size. Transactions can have different sizes depending on the number of inputs, outputs and the size of the scripts. A miner creates a block by filling this list and receive a fee as an incentive. He will then wants to put a maximum of small transaction with high fees.

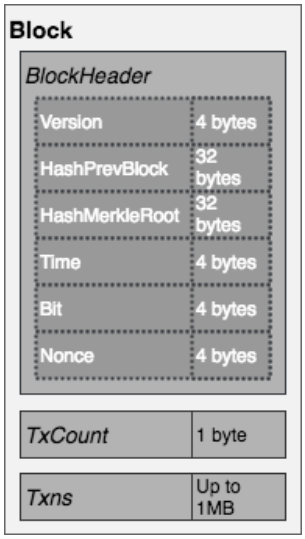

Fig. 3. Block data structure

A Block header is the main element of a block and allows

'b' + 32-byte block hash	**block index record**	-Block header. -Height of the block. -Number of transactions. -Block Validation State. -Number of the blk*.dat. -Number of the rev*.dat.
'f' + 4-byte file number	**file information record**	-Number of blocks stored in the block file. -Size of the block file ($DATADIR/blocks/blk*.dat). -Size of the undo file ($DATADIR/blocks/rev*.dat). -Lowest and highest height of blocks stored in the block file. -Lowest and highest timestamp of blocks stored in the block file.
'l'	**4-byte file number**	-Last block file number used.
'R'	**1-byte boolean**	-'1' if true reindexing.
'F' + 1-byte flag name length + flag name string	**1 byte boolean**	-'1' if true, '0' if false. -txindex
't' + 32-byte transaction hash	**transaction index record**	if txindex=1 -blk*.dat. -beginning of the block. -Beginning of the transaction from the beginning of the block.

Table 1: Block Index database

for the node to do simple validation on the block. It is composed of six fields. The first one contains the version number that indicates the validation rules version this block follows. The second one contains the hash of the previous block or parent block. This hash is SHA256(SHA256(BlockHeader)) stored in the big endian format, meaning that the hash starts with multiple zeros. The third field contains the hash of the merkle root. The merkle tree is created based on the transactions list. It's a binary tree where the leaves (transactions) are hashed and concatenated recursively until reaching the root. Even the smallest manipulation will change the root guaranteeing the immutability of the transactions and the block. The fourth field contains the timestamp using the standard UNIX time. The fifth field contains the difficulty parameter that tells the miner how many zeroes the hash must have from the start. The last field of the block header contains the nonce, a random number used to find a hash under at the correct difficulty.

A miner will find the correct hash by modifying 3 values: the timestamp, the nonce and the merkle tree. As the hash rate of individual device increases, the miner realizes that changing only the timestamp and the nonce is insufficient. A miner can create different hash for the coinbase, using the free space inside the coinbase transaction input. The impact of this, generates a new merkle root, thus changing the possible hash of the header. We remark that the block header hash isn't sent to the network. It's unnecessary because all the components permitting to verify the hash are inside the block header. And a node tries to rehash using the block header, if the hash is at the correct difficulty it means that it's at least a valid block header.

c' + 32-byte transaction hash	**unspent transaction output record for that transaction**	-Version -Is a coinbase -Height block that contains the transaction. -Outputs of that transaction that are unspent. -ScriptPubKey and amount for the unspent outputs.
B'	**32-byte block hash**	Block hash of the last unspent transaction output

Table 2: UTxO database

C. Databases

The bitcoin core software uses four databases [13]. The first one stores the blockchain in block files, the second one stores an index of the blockchain in a LevelDB database, the third one stores the unspent transactions in a LevelDB database, and the last one stores the modification of new block. Nodes need to store all the databases on the machine. It's mandatory as the databases are consulted when the node does the transaction verification.

1) Raw block data

The actual blockchain database should be accessible at any time for revalidation. It stores every valid block; blocks that are part of the blockchain, stale blocks and orphan blocks. Each file is 128MB large, in chunks of 16MB. Two nodes may depend on different databases, as only the longest branch is downloaded when a new node connects to the network. And when a block become stale is no longer propagated on the network. Some nodes won't ever display stale or orphan blocks in their database. The files contain raw data of the blocks. The filename nomenclature is blk*.dat.

2) Block index

The block index contains basic information about the blocks and where to find them in the raw block data. It stores every block index; blocks that are part of the blockchain, stale blocks and orphan blocks. This is key-value database [14] featuring six keys. The first one corresponds to block information that stores the whole block header, the height of the block, the number of transactions in it, the validation state of the block, the raw data block file number, which in turn stores this block and the raw undo data number corresponding. The second value stores information about raw data block files and contains the number of blocks in the file, the size of the file, the size of the corresponding raw undo data file, the lowest and highest heights of the blocks and the lowest and highest timestamps. The third key is the number of the last raw block data file created. The fourth key is a Boolean if set to 1 thus indicating that the blockchain is in the process of reindexin and that means the software is in the process of recreating both chainstate and index. The fifth key is where flags are stored. The flags can only have a Boolean value. For example a txindex flag set to 1 means that the main index stores a transaction index for each blocks. The last key is optional and only used if the flag txindex is set to true. It stores the raw block data file number where the transaction is stored, the offset at which the cursor is initially placed when

reading the block that contains the transaction, the offset indicating where the transaction is, starting from the beginning of the block. The main advantage of the block index is easing the validation process as the nodes can find the related information quicker than navigating through the whole blockchain.

3) Unspent Transaction Output (UTxO) or Chainstate

This database contains all the transaction hashes where outputs still need to be spent. This database is created as a result of the validation of blocks. When a node downloads the blockchain or a new block, it will validate all the new blocks. This process creates the UTxO. This is key-value database [14] featuring two keys. The first one contains the version of the validation rules that the transaction follows, either or not the transaction is a coinbase, the height of the block where the transaction is, the unspent outputs, the ScriptPubKey and the amount of those unspent outputs. The last key stores the block hash that contains the latest unspent output known by the database. Every block after this one won't be checked when validating transactions.

4) Raw undo data

The data is stored in files. Each one contains the serialized transactions outputs of the block and a checksum. The checksum is used when a block needs to be disconnected during a reorganization of the blockchain. Its role is to guarantee that the undo data weren't modified. Raw undo data block and raw data block are twin. The filename nomenclature rev*.dat the * is the same as the corresponding blk*.dat.

IV. VALIDATION METHOD

In this section, we explain the validation process inside the Bitcoin. First, we will talk about the transaction and the criteria that are checked, and then we will look at how a block is linked to the blockchain.

A. Transaction creation

Creating a transaction implies that the sender's public address already has an output proving his ownership. This output can be a coinbase or an output from a transaction. Contrary to the traditional system of transferring assets, a Bitcoin user doesn't have an account, but has instead a set of private keys linked to public keys. Which means that transactions are between keys not person. Those actions are done with the wallet functions. It is possible to create a transaction without the help of a tier software but it needs a lot of effort [15]. Also a wallet is convenient for the user, as it stores locally the information about the outputs owned by him and shows the total balance of bitcoin spanning over multiple addresses.

First the wallet consults the UTxO to retrieve unspent transactions outputs on which the cumulative amount is at least superior to the amount to be sent. The sender then adds the output TxID and the index as an input followed by his ScriptSig for each output. The input can contain outputs issued to different private keys. The output is then added to the transaction. A transaction can contain multiple outputs each of which needs to be associated with a public key. This key is the destination of the output and determines who has the right to spend the output. It's worth noting that if the sender has more bitcoin than the spent amount, he needs to create an output called the "Change output" of which he is the recipient. If the sender doesn't create the Change output the bitcoin amount remaining will go to the miner as a fee for the transaction. Either way the sender has to put a fee on his transaction. This fee will serve as an incentive to motivate the miner to place the transaction inside of a block, as the fee is a direct reward for him. Miners tend to choose the transactions with the highest fee. The higher the fee the faster the transaction will be added to the blockchain. As of now (June 13,2017) the fee is 420 satoshis/byte, the median transaction size is 226 bytes, resulting in a 94,920 satoshis (0,00094920 bitcoins) per transaction [16].

B. Transaction validation

A transaction is sent to the Bitcoin network. Where it's broadcast by all nodes upon clearance of its validity. The transaction follows different criteria that are described below. These criteria set the consensus rule for adding data inside the blockchain. The majority dictates what rules the minority needs to follow. The Bitcoin Core developers set the rules. Currently as it's the most popular implementation on the network, however, a new set of rules can be pushed by a users majority using another software implementation. This new rule is thus set for the consensus.

We divide the validation process into four steps, and we will consider that the node is honest as a malicious node might propagate transactions, which aren't valid.

1) Version verification

The node checks if the version of the transaction is standardized and implemented in the software.

As of now (06/2017) the standard version code is 2. And all other code will be considered non-standard and the transaction will be discarded by nodes running the Bitcoin Core developers set rule.

2) Simple verification

Multiple verifications occur during this step, there are inexpensive for the node as they need little computational power. Because the accesses to the database are short and fast, they are energy efficient.

At first the node checks if there is an input in the transaction,

then for each input:

- It checks if the input isn't already referenced in the UTxO as an input. The database chainstate will be consulted.
- It checks if the input isn't already saved in the blockchain as an input. The database index will be consulted.
- If the input is a coinbase, it checks its maturity. There must be a difference of 100 blocks between the current height and the height of the coinbase creation.
- It checks if the amount neither negative nor overflowing (not superior to 21 million).

The node then checks if the total input amount is superior to the total output amount. Last it checks that the fee is positive but less than 21 millions bitcoins.

3) Signature verification

This verification is the most demanding in computational power. As the node executes the scripts, first the ScriptSig that pushes data onto the stack, then the ScriptPubKey are executed. Those scripts compose the inputs of a transaction. The node retrieves the ScriptBupKey of the inputs from the database and are stored in the Chainstate.

1) P2PKH

The script will compare the public key from the ScriptSig to the public key hash. If they are identical then it checks if the signature matches public key. The script returns TRUE at the end of its execution if the signatures match. If it returns FALSE, then the transaction is discarded.

2) P2SH

The script compares the redeemScript hash present in the ScriptSig to the one in the ScriptPubKey. If the hash matches the redeemScript will be executed.

4) Locktime verification

The node checks if there is enough time between the block containing the input and the current UNIX time. The blockchain can validate this up to 2 hours in advance.

C. Block manipulation

In this section we will explain how the blocks are attached to the blockchain and how the reorganization occur when a fork is solved. A block has different states of validation from 1 to 5 [17], this state is stored in the block index database and expresses how much trust you can put the block:

1. Valid_Header, the block header has a valid proof of work, the number of transactions is between 1 and Max, the timestamp isn't in the future.
2. Valid_Tree, all parent headers are valid, the difficulty match the current one, the timestamp is superior to the median of the previous blocks, all its parents are at least at this level of validation.
3. Valid_Transactions, there is only one coinbase transaction and it's the first one, the size of the coinbase input script length is between 2 and 100 bytes, all the transactions are valid, no duplicate of TxID with the UTXO, the signature scripts are valid (only pushing data to the stack), the size of the block is valid (under 1MB), the merkle root is valid, all its parents are at least at the Valid_Tree state.
4. Valid_Chain, inputs superior to outputs, no double spends, coinbase output ok, if there is coinbase spent then check if the maturity is more than 100 blocks, all its parents are at least at the Valid_Chain state.
5. Valid_Script, all the signatures and scripts pairs are valid, all its parents are at least at the Valid_Chain state.

1) Connecting and validating a block

Connecting a block to the blockchain validating a block are two different instances. If a node receives an orphan block, it has to make sure that this block is valid and saves it in case the missing parent comes back afterward. The standard procedure is as follows:

Upon receiving a block a node validates it. First, it runs a verification on the header by checking the proof of work, and then, if the merkle tree corresponds with the transactions. If the parent is known, the node proceeds to further validate the block, if not however the block will be kept, waiting for its parent. In both cases the block is saved on the disk both in the raw data and the block index. The validation continues for blocks with parents and the node now checks the transactions inside the block. The procedure is the same as the transaction validation except that it's executed for every transaction inside the block. Once the block reaches the Valid_Script state, it's then eligible to become the next tip of the blockchain, and the node connects the block to the blockchain. The different databases are updated following the current state of the blockchain.

2) Disconnecting a block

When a fork occurs the network will resolve this problem by electing the longest branch as the legitimate one, thus, the blocks in the losing branch become stale. We should assess if the transactions inside of it are in the longest branch or not. The node uses the raw undo data file to remove the associated blocks. It unspends all transactions inside the block in reverse order (in case of nested transactions), removes the outputs in the UTxO and restores the inputs into the UTxO. All

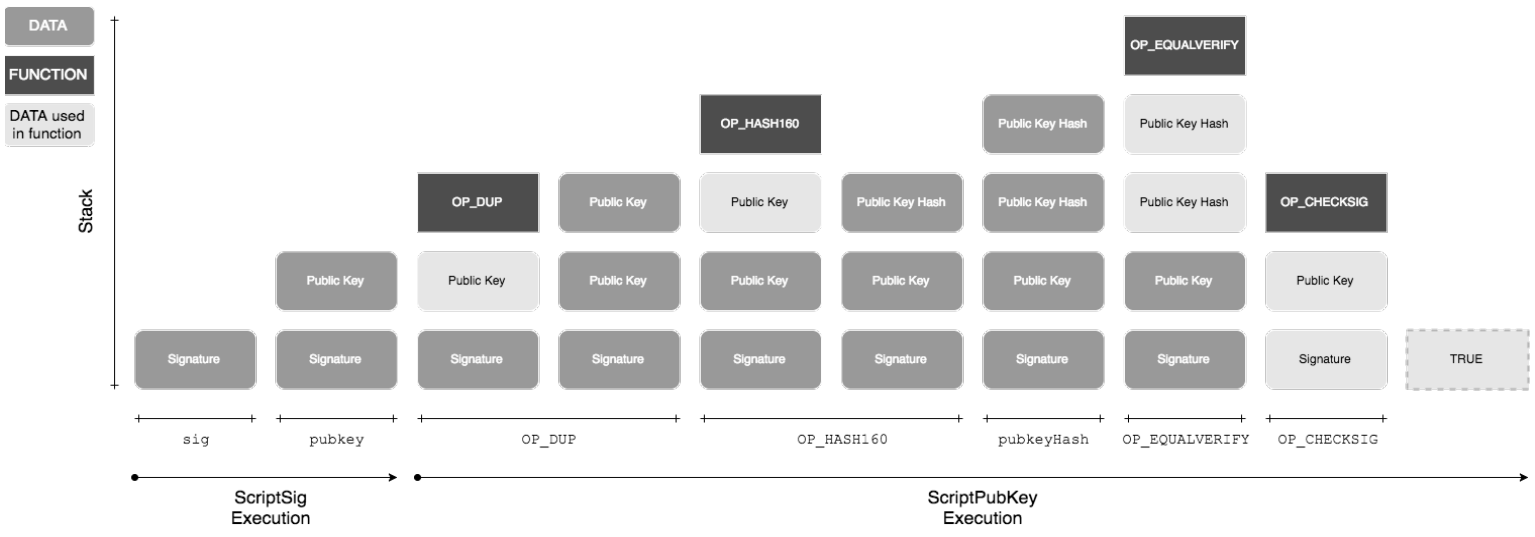

Fig. 2. P2PKH script execution; show the evolution of the stack during the signature verification.

transactions sent to the mempool are revalidated in case they are already added in the longest chain. It's important to remember that even though the block isn't part of the blockchain, it is still saved in the databases and on the disk.

V. Future work

In this section, we will present you with the results of our research to build a test bench. To emulate the bitcoin environment we identify 3 occurring events: new transaction, new block and fork. The last two are well researched by [12], but knowledge there isn't a model for new transactions rate.

We needed the transactions rate per second data to feed in the mempool. We used a dataset from [18] spanning from August 14, 2016 to August 14, 2017. We observed from this, that the rate is approximated by a lognormal distribution. The parameters are σ=0.29731261148742899 and μ=1.04495844426, see figure 3. With X_t a random variable of the number of transactions per second, it has the following probability:

$$P_t = \frac{1}{X_t \sigma \sqrt{2\pi}} e^{\frac{(lnX_t - \mu)^2}{2\sigma^2}}$$

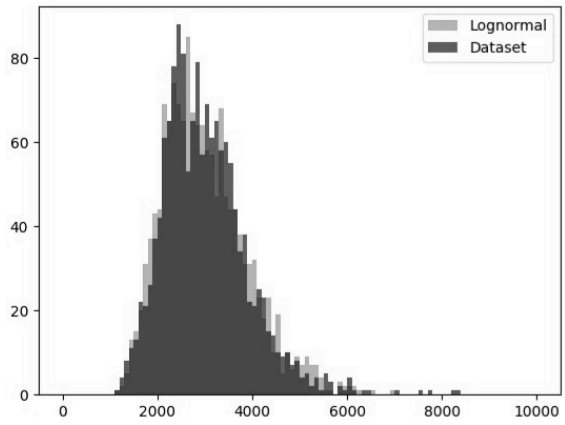

Fig. 3.Comparison between a lognormal distribution and the dataset

After analyze this results, we realized the need to build a test environment. We choose to use the container technology as it host the nodes to gave us more control on the specifications and it was easier to deploy than traditional hardware or virtual machines. We used Docker to run the Bitcoin core client and we wrote python scripts using the RPC commands to remotely control the nodes. We also configured the nodes so they used the ZeroMQ function, in order to monitor incoming transactions and blocks on each node. This work is still in progress and will be completed at later date.

VI. Conclusion

In this paper we offer an analysis of the validation process in the Bitcoin blockchain. We saw that a node consults actively the blockchain during the validation. The expensive element of the validation is the verification of the script, which needs more computational power than looking up a database. Still the Bitcoin is slow enough that every node has the time to do enough verifications before receiving a new block. The protocol ensures that only valid blocks are added to the blockchain. We also overview how the process is tied to the consensus and how it is designed to help the synchronization between nodes. This work is the foundation for our future research on using blockchain in the manufacturing industry.

References

[1] J. Evans, "Blockchains are the new Linux, not the new internet," *TechCrunch*.

[2] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wirel. Pers. Commun.*, vol. 58, no. 1, pp. 49–69, 2011.

[3] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: challenges and solutions," *ArXiv Prepr. ArXiv160805187*, 2016.

[4] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the Security and Performance of Proof of Work Blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2016, pp. 3–16.

[5] A. Kiayias and G. Panagiotakos, "Speed-Security Tradeoffs in Blockchain Protocols," 2015.

[6] S. Popov, "The tangle," *Available Electron. Httpiotatoken ComIOTA Whitepaper Pdf*, 2016.

[7] *bitcoin: Bitcoin Core GitHub*. Bitcoin, 2017.

[8] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.

[9] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore, "Game-Theoretic Analysis of DDoS Attacks Against Bitcoin Mining Pools," in *Financial Cryptography and Data Security*, 2014, pp. 72–86.

[10] A. Laszka, B. Johnson, and J. Grossklags, "When Bitcoin Mining Pools Run Dry," in *Financial Cryptography and Data Security*, 2015, pp. 63–77.

[11] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." 2009.

[12] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*, 2013, pp. 1–10.

[13] "Bitcoin Core 0.11 (ch 2): Data Storage - Bitcoin Wiki." [Online]. Available: https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_2):_Data_Storage. [Accessed: 02-Jun-2017].

[14] "What are the keys used in the blockchain levelDB (ie what are the key:value pairs)? - Bitcoin Stack Exchange." [Online]. Available: https://bitcoin.stackexchange.com/questions/28168/what-are-the-keys-used-in-the-blockchain-leveldb-ie-what-are-the-keyvalue-pair. [Accessed: 02-Jun-2017].

[15] "Bitcoins the hard way: Using the raw Bitcoin protocol."

[16] "Bitcoin Fees for Transactions | bitcoinfees.21.co." [Online]. Available: https://bitcoinfees.21.co/. [Accessed: 05-Jun-2017].

[17] *bitcoin: Bitcoin Core. chain.h l.125*. Bitcoin, 2017.

[18] "Transaction Rate," *Blockchain.info*. [Online]. Available: https://blockchain.info/transactions-per-second. [Accessed: 08-Sep-2017].