Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Help the Bitcoin Community Reach Consensus

Semester Thesis

Alexandros Filios

afilios@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Prof. Dr. Roger Wattenhofer, Dr. Christian Decker

February 2, 2016

Abstract

The recent discussion about changes to the Bitcoin protocol have shown that the Bitcoin community is unable to reach timely decisions with a transparent process. Despite starting as a trivial change, removing an artificial limit to the maximum number of transactions, the discussion quickly degenerated into heated discussions about non-issues, name-calling and speculation, that to date did not produce a working long-term solution and is unlikely to do so, given the large number of competing proposals. It is clear that the Bitcoin community needs tools to guide the decision process into a more productive direction. The first step in the decision process is to publicize the various options, who endorses which option and why. To this end we want to build a reputable tool that can be used by the members of the Bitcoin community to express their goals in a pseudonymous way and weight their opinions based on the stake they have in the ecosystem.

Contents

Chapter 1

Introduction

Since its inception, Bitcoin[1] – an online payment system published by Satoshi Nakamoto in 2008 and launched as an open source software in 2009 – has markedly grown in popularity and use with more and more people embracing its concept and participating either actively (miners) or as simple consumers of the services (users). One of the biggest virtues of Bitcoin that has led to its gaining more mindshare in the general population with each passing day, capturing the attention of people, governments and businesses around the world is its democratic character.

For the first time in history we have the ability to guarantee that with the appropriate coordination and and design each individual can have a voice and that this voice will not be impersonated. We can guarantee one vote per person and we can guarantee that it is that person casting the vote. This gives to all of the participants of the Bitcoin ecosystem the right to decide in common over important issues concerning the system such as the customization of the mechanism parameters, rather than leave such decisions up to a central authority.

The objective of this work is to investigate alternatives in order to give participants in the community the possibility to vote and decide over a currently hot topic in Bitcoin: the block size.

The structure of the thesis will be as follows: At first we will provide an insight into the basic concepts of the Bitcoin protocol upon which our analysis will be based. Afterwards we will define the problem, its parameters, the goal and the scope of the problem. Next we will present and discuss the alternatives to solve the current problem, as well as elaborate a complete solution including the implementation details. Next, an evaluation of the system will follow with some representative scenarios and graphical representations of the system performance and eventually we will summarize the main results and conclusions of the work.

[1] The word "bitcoin" (with a small *b* letter) refers to the coins, while the term "Bitcoin" with a capital *B* refers to the protocol itself.

Chapter 2

What is Bitcoin

Bitcoin is a peer-to-peer protocol; users can transfer Bitcoin value directly without an intermediary. *Transaction* is a data structure describing a transfer that is broadcast to the network. It typically references previous transaction outputs as new transaction inputs and allocates all input Bitcoin values to new outputs. Standard transaction outputs nominate *addresses* and the redemption of any future inputs requires a relevant signature [1].

Each participant in the Bitcoin environment has a collection with one or more key pairs: the private key components are only known by the owner and stored in a file called *wallet*, while the matching public keys derive directly the Bitcoin addresses of the user. This ability to keep multiple addresses per user constitutes the immunity mechanism of Bitcoin against tracking and contributes to the system's privacy.

Transaction data is permanently recorded in *blocks*. Blocks are organized into a linear sequence over time (also known as the *block chain*). New transactions are constantly being processed by *miners* into new blocks which are added to the end of the chain and can never be changed or removed once accepted by the network [1].

When a transaction is included in a block that is published to the network, it is said that the transaction has been mined at a depth of 1 block. With each subsequent block that is found, the number of blocks depth is increased by one. To be secure, a transaction should not be considered as *confirmed* until it is a certain number of blocks depth. Otherwise, a transaction may be included in a block whose inputs had already been spent in another transaction in the blockchain (double spending).

Mining is intentionally designed to be resource-intensive and difficult for the miners to produce, so that the number of blocks found each day by miners remains steady. This *proof of work* included in each block satisfies certain requirements and is easy for peers on the network to verify so as to confirm that a block has been successfully mined. The computational effort lies in finding the SHA-256 hash of a block's header that will be lower than or equal to a certain

globally set target (that changes regularly every 2016 blocks), or equivalently, that will start with a certain number of zeros. This happens because the probability of calculating a hash that starts with many zeros is very low, therefore many attempts must be made [1].

Bitcoins are created as a reward for payment processing work in which users offer their computing power to verify and record payments into the blockchain. This activity is called a *mining* and miners are rewarded through

- *transaction fees*: bitcoins only optionally included but usually expected with any transfer of bitcoins from one address to another
- the *coinbase transaction*: a special transaction, unique per block, that has no inputs and grants a limited amount of (newly created) bitcoins to the miner that mines the block and has hence the right to generate such a transaction.

Besides being obtained by mining, bitcoins can be exchanged for other currencies, products and services [1].

2.1 Block Structure

For the needs of this project, we will have to insert some information in the block chain, which means that the already existing fields of the block format have to be exploited. To this end, in this section we will take a closer look on the block structure and how the various fields are currently used. Following this we will be able to explore which of them can be further modified, that is, be given an external non-default value, obtaining this way some additional meaning without of course clashing their original purpose – thus invalidating the block information.

As can be seen in Table 2.1 a block consists of two main parts: the header and a list of transactions. The following have their fields defined by the miner and seem to be the most useful for our purpose:

- **Coinbase transaction:** It is created by miners, and there is one coinbase transaction per block. While regular transactions use the inputs section to refer to their parent transaction outputs, a generation transaction has no parent, and creates new coins from nothing. The coinbase can contain any arbitrary data.
- **Version:** The version field is split into two parts. The first is allocated for the actual version, while the remaining is to do flagging. The version field is 4-byte long (32 bits), of which the 3 most significant bits are set to `001`.

Focusing now in a single transaction, as depicted in Table 2.2, they consist of *inputs* and *outputs*. The former refer to records which reference the funds to be spent from other previous transactions, while the latter denote records which determine the new owner of the transferred bitcoins, and which will, in their turn, be referenced as inputs in future transactions as those funds are respent. Of particular interest for our needs are the fields below:

- **Lock time:** This field refers to the point until which the transaction is locked and allows signers to create time-locked transactions which will only become valid (and hence enabled to be added into a block) in the future, giving the signers a chance to change their minds. The field is interpreted as follows:
  - `0`: Not blocked. The field in this case is irrelevant.
  - `< 500,000,000`: Block number at which this transaction is locked
  - `>= 500,000,000`: UNIX timestamp at which this transaction is locked
- **Sequence:** A field included in each transaction input and was intended for *replacement*, a currently disabled feature. How it would work is:
  - We send a transaction with a `Locktime` in the future and a sequence number of 0. The transaction is then not considered by the network to be "final" and it can't be included in a block until the specified `Locktime` is reached.
  - Before the `Locktime` expires, we can replace the transaction with as many new versions as we want. Newer versions have higher sequence numbers.
  - If we ever want to lock the transaction permanently, we can set the sequence number to `UINT_MAX`. Then the transaction is considered to be final, even if the `Locktime` has not been reached.

  Since replacement is not used currently, all transactions Bitcoin Core creates have by default *Sequence* = `UINT_MAX` making the transaction final and hence disabling the lock time. Thus, in this case the lock time field can be set to any value (by default in Bitcoin Core *Locktime* = 0) having any effect on the transaction.
- **Scripts:** A script is a list of instructions recorded with each transaction that describe how the next person wanting to spend the bitcoins being transferred can gain access to the. Scripts are found in both inputs – as `scriptSig` – and outputs – as `scriptPubKey` – of a transaction. Scripting provides the flexibility to change the parameters of what's needed to spend transferred bitcoins [1].

	Description	Comments
Header	`version`	Block version information
	`prev_block`	The hash value of the previous block this particular block references
	`merkle_root`	The reference to a Merkle tree collection which is a hash of all transactions related to this block
	`timestamp`	A timestamp recording when this block was created
	`bits`	The calculated difficulty target being used for this block
	`nonce`	The nonce used to generate this block
	`txn_count`	Number of transaction entries
TXs	Coinbase transaction	
	Transaction 1	
	Transaction 2	
	...	
	Transaction `txn_count`	

Table 2.1: Block structure

2.2 Consensus and Natural Forks

Each full node – a program that can fully validate transactions and blocks – in the Bitcoin network independently stores the whole block chain, in contrast to lightweight nodes that need not maintain a full copy of the chain. When several nodes all have the same blocks in their block chain, they are considered to be in *agreement* [2].

Nevertheless, under certain network conditions, two different blocks may be found at close time instances by two different miners. Then both of them will have valid and legitimate blocks, and neither will have a reason to toss it out. Then both will broadcast their newly-validated block out to the network and some people update their block chain one way, and others update their block chain the other way, creating two different heads. Technically this is a *natural fork* in the block chain [3].

To resolve this problem, the peers in the network keep track of both forks, but only working to extend whichever fork is longest in their copy of the block chain at any given time. As a result, the miners working on the fork that extends more slowly will eventually abandon it and start working on the other fork. Of course, any still-pending transactions in the shorter fork will still be pending in the queues of the miners working on the other fork and so all transactions will be validated in the end.

Description			Comments
version			Transaction data format version
tx_in count			Number of transaction inputs
tx_in[]	tx_in[1]	description	comment
		prev_out_hash	The hash of the referenced transaction
		prev_out_idx	The index of the specific output in the transaction
		script_length	The length of the signature script
		sig_script	Computational script for confirming transaction authorization
		sequence	Transaction version as defined by the sender. Indended for "replacement" of transactions when information is updated before inclusion into a block.
	tx_in[2]	...	
	...	...	
	tx_in[tx_in count]	...	
tx_out count			Number of transaction outputs
tx_out[]	tx_out[1]	value	Transaction value
		pk_script length	Length of the pk_script
		pk_script	Usually contains the public key as a Bitcoin script setting up conditions to claim this output.
	tx_out[2]	...	
	...	...	
	tx_out[tx_out count]	...	
lock_time			The block number or timestamp at which this transaction is locked.

Table 2.2: Transaction structure

This process ensures that the block chain has an agreed-upon time ordering of the blocks. As a rule of thumb, a transaction is considered confirmed when it is part of a block in the longest fork and at least 5 blocks follow it in the longest fork, or equivalently, the transaction has 6 confirmations.

Chapter 3

Evolving the Protocol

In our attempt to enable the further evolution of the Bitcoin protocol and the enhancement of its functionalities, it is essential that changes be made on the way the different fields of a block are read and interpreted by the network nodes. It comes as an immediate consequence that these nodes have to adjust the block validation rules they adhere to, that is, the rules that a full node follows in order to determine whether a block is valid or not.

Since Bitcoin is a distributed protocol, it is generally not expected that these modifications will be applied simultaneously and in a coordinated fashion by all peers and therefore it can often happen that different full nodes follow different block validation rules until the changes are propagated, accepted and finally adopted throughout the peer network. Using the backward compatibility as the differentiating factor, all protocol changes can eventually fall into two main categories, which we will study in the following: the *softforks* and the *hardforks*.

3.1 Soft vs. Hard fork

A *hardfork* is a change of the Bitcoin protocol that is not backwards-compatible, i.e., older client versions would not accept blocks created by the updated client, considering them invalid. This can create a blockchain persistent fork – as opposed to the natural forks – when nodes running the new version create a separate blockchain incompatible with the older software. After a hardfork, all users are required to upgrade. Any iteration to bitcoin which changes the block structure (including block hash), difficulty rules or increases the set of valid transactions is a hardfork.

A *softfork* is a change to the bitcoin protocol wherein only previously valid blocks/transactions are valid. Since old nodes will recognize the new blocks as valid, a softfork is backward-compatible. This kind of fork is enforced once the majority of the miners upgrade to the new rules.

New transaction types – that would otherwise be implemented as hardforks

– can often be added as soft forks, requiring only that the participants (sender and receiver) and miners understand the new transaction type. This is done by having the new transaction appear to older clients as a special form of transaction (so-called pay-to-anybody" transaction) and getting the miners to agree to reject blocks including these transaction unless the transaction validates under the new rules [1].

3.2 The Ecosystem

There can be numerous categorizations of the participants in the Bitcoin ecosystem, depending on the viewpoint and the case under study. In the context of the present work, we will consider that the different interests of the various involved parties in the bitcoin community can fall into three main categories.

The two first categories are two different views of the *users* of the protocol. These are the people making use of Bitcoin in order to place payments, that is, to send bitcoins to each other over the network. The smallest unit of their activity is a single transaction. However, since Bitcoin payments can be used for different kinds of activity, we expect that a finer categorization of the users into *stakeholders* and *high-traffic users* would be more appropriate in our attempt to define their incentives and interests. The third category will be that of the *block miners*. In more detail:

- **Stakeholders (SH):** These are the principal value holders and long-term investors, even if they do not present considerable transaction activity.
- **High-traffic users (HTU):** These are the users responsible for the highest activity in terms of number of transactions, namely the users that create the biggest amount of traffic. Many transactions may consist of small amounts of bitcoins sent back and forth to the same wallets/users. There are quite a few cases where such transactions are indispensable, such as *SatoshiDICE*[1] and *BitPay*.[2]
- **Miners:** These are responsible for the block creation, i.e., the addition of the transaction records that the users have announced to the network into the Bitcoin's public ledger of past transactions through mining [1].

[1] In SatoshiDICE bets, a Bitcoin transaction is made to one of the static addresses operated by the service, each having differing payouts. The service determines if the wager wins or loses and sends a transaction in response with the payout to a winning bet or it returns a tiny fraction of the house's gain to a losing bet [1].

[2] BitPay payments try to mitigate the risk of bitcoin price volatility by allowing the merchant to accept bitcoin and immediately converting it to US Dollars, Euros, or the currency of the merchant's choice [4].

Although it is straightforward to distinguish the miners from the users, this is not the case between SHs and HTUs. This happens because of the inherent property of Bitcoin to prevent user-activity tracking, that is, we cannot record the activity and profile a specific user by just observing the blockchain transactions, due to the mechanisms that the protocol has introduced to protect privacy, e.g. the creation of a new address for each payment (that however belongs to the same user wallet).

However, it is not our intention to cluster the users, but rather consider the characteristics of their activity, e.g. value transferred in a single transaction, and weigh them accordingly depending on the view we consider, that is, the same user will eventually impact both views (SH and HTU), but more in one and less in the other, depending on their behavior.

3.3 Problem Statement and Goal

Major policy changes in an open source project are messy. What would be a closed door policy discussion in a central bank, is now instead held out in the open, with all parties airing their opinions in an open forum.

One of the main reasons for the popularity of Bitcoin is the fact that it can support an instant, secure, decentralized, trustless and egalitarian currency system where all parts of the ecosystem can express their opinions on the current policies as well as any proposed changes. It is these very characteristics that can make it possible to move away from a centralized management, improving bitcoin's governance by removing a hardcoded policy control from the software.

Nonetheless, although the Bitcoin offers these enablers, such features are not natively supported by the protocol. To this end the main goal of the present project is to create a platform that will allow participants to cast their votes regarding a certain proposal in a standardized way and offer them an appropriate way to insert this information into the blockchain. This will help the system eventually reach consensus, while not violating the following important conditions:

- Our system should *avoid hardforks* which would allow backwards compatibility as well as an easier and more immediate deployment on the existing infrastructure.
- Users should be able to vote *maintaining their anonymity* that the Bitcoin protocol guarantees them through its mechanisms and are responsible to a great extent for its popularity.

Chapter 4

Case Study: The Block Size Debate

Although the present work covers a more generic topic, that of the ecosystem need to vote, we will put these ideas into practice and demonstrate them in a case study, concerning one of the most crucial and debatable issues in the Bitcoin community: the *block size limit*.

Absent a speed limit, of sorts, attackers bog down the entire system. In the early days of bitcoin, when the dollar and processing cost to flood the system was low, originator Satoshi added a limit on the total number of transactions validated every 10 minutes.

When the number of transactions in the past 10 minutes exceeds this speed limit, users start to bid for block space by including fees in their transactions. Bitcoin transactions are prioritized based on the transaction fee attached. The higher the fee paid by the user, the more likely their transaction will be prioritized before a less-paying user [5].

4.1 Incentives

4.1.1 Arguments directly affecting categories

Having already described the three categories in the Bitcoin network that we would like to consider under the perspective of this project, we can now recognize their incentives behind pressuring towards bigger or smaller block sizes. Distinguishing the two cases, we have the following observations to make:

- **Smaller blocks:** In general, fast confirmation does not have to do with the block size (although a small block means that less transactions are in the block and hence the block can go around the network more quickly, thus reaching the same level of consensus in a considerably less amount of time). This refers to the first approx. 20 sec, until many nodes have

received the block, avoiding the danger of a fork. A small block could take only 10 sec to be received by the same amount of nodes. Then the rest of the network will stop competing against this one and therefore we will have less double spends (faster propagation speeds).

Also, small blocks will require higher fees for fast confirmations because now less transactions will content for a place in the block.

As a consequence the motives of the three categories will be as follows.

- **High-traffic users:** It will no longer be cheap to generate transactions at a high rate.
- **Stakeholders:** Bitcoin may look unattractive to new users with high fees, hence stopping or reversing global adoption, investment, development, support and centralization. As a consequence, this may take its toll on the popularity and value of bitcoin, resulting in a decrease on the value already owned. The increased transaction fees per se should remain neutral for users with high capitals and a moderate transaction traffic generation.
- **Miners:** Fees will not be zero. Instead, a low block size limit encourages higher transaction fees to incentivize miners ("let a free market develop"). This is eventually a necessity in order to incentivize miners and secure the mining ecosystem.

- **Bigger blocks:** Although, as explained above, less forks will be caused with smaller blocks rather than with bigger ones, the fact that we can include much more transactions in the same blocks, will allow for more and cheaper transactions per time unit. However, the incurred network latency will increase the possibility of rejected transactions.

  Each group will have then the incentives presented below.

  - **Miners:** Incentivized through higher transaction fees. Since more transactions are now included int he block, more fees are to be collected by the miners [6].
  - **High-traffic users:** Lower fees will allow them to afford increased transaction-creation activity.
  - **Stakeholders:** Lower fees but higher transaction rejection possibility.

4.1.2 Arguments indirectly affecting categories

Apart from the incentives illustrated above, some additional arguments are to be pointed out and taken into consideration that nevertheless are not directly linked to the interests of some specific group (of the ones we are examining at least). Such points include the following:

Incentive	Small block	Big block
High-traffic users		
High transaction fees		✓
Stakeholders		
Lower transaction fees		✓
High rejection probability	✓	
Bitcoin popularity		✓
Miners		
Free market	✓	
More transaction fees	✓	
Miscellaneous		
Node maintenance costs	✓	
Centralization	✓	
New application areas		✓

Table 4.1: Summary of incentives towards a smaller or bigger block size.

- **Centralization:** Since larger blocks lead to less miners running full nodes, this leads to centralized entities having more power, which makes Bitcoin require more trust, which weakens Bitcoin's value proposition [1]. Bitcoin is only useful if it is decentralized because centralization requires trust.
- **Node maintenance costs:** Moreover, the network will be relieved due to the lower traffic load that a small block size would bring to the network, i.e. consider a block of 1 MB and a block of 2 MB that have to be flooded after their generation to all of the full nodes which will now be more expensive to operate (more processing power). This will also boost the performance of lightweight nodes that receive a block only after request. These arguments promote the use of small block sizes.
- **New application areas:** Bigger block sizes will leave space for extensions like Mastercoin, Counterparty, etc.

Summarizing the results of the above analysis, we conclude to table 4.1.

4.2 BIP100 and BIP101

So far, it has already been clear that the current block size limit is too low and will delay growth and lead to user pain and expense if not increased. To solve this problem, two schemes have already been proposed for the transition to

bigger block sizes, with appropriate mechanisms to regulate each phase of this transition.

- **BIP 100:** In this scheme, miners vote by suggesting directly their desired block size. Votes are evaluated by dropping bottom 20% and top 20% and then the most common floor (minimum) is chosen. Moreover, the increase or decrease of the block size may not exceed 2x in any one step.

  This procedure introduces friction into the block size increase process making it changeable, yet giving participants in the system sufficient time to observe system behavior and change course, ultimately moving towards a system where the market decides the best block size [5].

- **BIP 101:** This BIP proposes replacing the fixed 1 MB maximum block size with a maximum size that grows over time at a predictable rate. More specifically, the maximum allowed size of a block on the main network shall be calculated based on the timestamp in the block header.

  The maximum size shall be 8 MB on January 11th 2016 and shall double every 2 years until January 6th 2036, where the maximum size will stop increasing at 8.192 GB. The maximum size of blocks in between doublings will increase linearly based on the block's timestamp [7].[1]

The main difference of the voting scheme of the two proposals lies in the fact that in the first case (BIP 100), the miners vote directly for a block size, while in the latter (BIP 101) they vote only for or against a predetermined action plan.

Nevertheless, the aforementioned schemes only consider the votes of the block miners, hence leaving no space to users to express their opinion, although they are still stakeholders of the ecosystem.

4.3 Scope

The tasks that belong to the scope of this work consist principally in the following points:

- Comparison and evaluation of alternative voting mechanisms for users and proposal of a solution.[2]
- Vote parsing, storing and counting for all three categories (as displayed in Figure 4.1).

[1] The doubling interval was chosen based on long-term growth trends for CPU power, storage, and Internet bandwidth. The 20-year limit was chosen because exponential growth cannot continue forever.

[2] Similar mechanisms are already defined for the miners in the BIP100 and BIP101 proposals, as well as adopted by many miners in the mainnet, thus we need not implement them again.

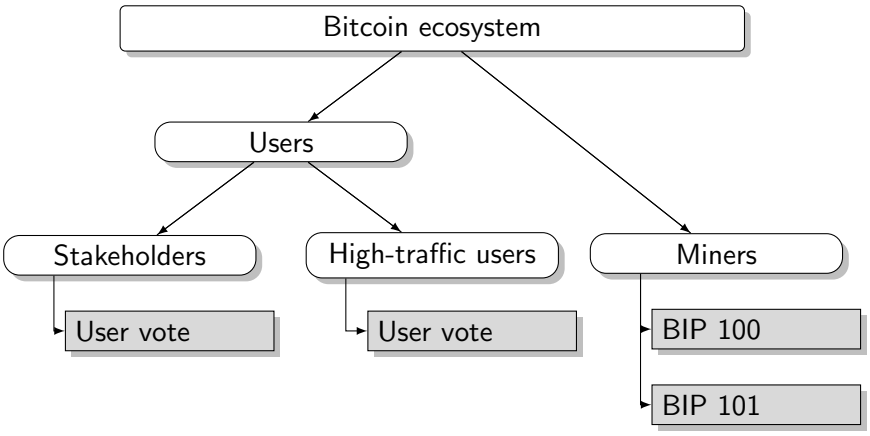

Figure 4.1: Different vote types encountered in our system.

- Proposal of the suitable observable metrics to allow for an appropriate weighting of the votes.
- Result extraction and evaluation according to these metrics.

Chapter 5

Implementation

Since each of the parties that form the Bitcoin community is not involved to the same extent and in the same manner, their preference (vote) should be given a different weight factor, that corresponds to their activity and the influence they have on the function of the network. To this end, we will define three different metrics to determine the weight of the votes for each of the categories explained so far.

5.1 Stakeholder Metrics

What distinguishes the behavior of a stakeholder is the fact that they hold bigger amounts of bitcoins, although the number of the transaction is markedly lower than that of the high-traffic users. As a consequence the weight that should be given to their vote should be *analogous to the transferred values.*

To this end, the weight value we will use for the metrics will now have to be proportional to the *unspent outputs* that each transaction comes with. This means that each transaction comes with a set of one or more outputs to other users (or the same user) and a vote. The weight of the vote will be proportional to the sum of the outputs that have not been referenced by an input so far. At the time of its creation the inputs equal the outputs (minus the transaction fees) and the vote has the maximum weight. Once one of the users that receive one of the outputs decides to use their output (the money transferred to them) in a transaction – that may be accompanied by a vote as well – the previous vote loses the corresponding amount of weight which is transferred to the new user, even if the receiver does not make use of this weight by not including any vote in their transaction. Therefore the votes of a user that has given all their bitcoins (and they have also been spent by the corresponding receivers) will have zero multiplier, or equivalently, will no longer influence the total vote results.

5.2 High-Traffic-User Metrics

What characterizes this category is the amount of transactions they produce. Therefore the metric should be proportional to the *produced traffic*, i.e., the number of transactions they transmit.

The problem in this case is that this value is constantly increasing over time (unlike e.g. the metric chosen for the stakeholders). For this reason, we can use the following weighing strategy: Choose a window of blocks, that is, a minimum block height up to the head of the chain, that will be considered for the calculations, so that the result is kept up to a certain limit.

5.3 Miner Metrics

Since the value of the transactions included in a block that a miner created does not say much about their influence and importance, we should use a metric more representative of the resource share they hold in the network, as well as their investment. One such metric would be the *absolute number of blocks* they have created so far, since that is directly related to the time they have been active so far, as well as the probability with which they can mine new blocks and hence the equipment they have invested in. Such a metric would be more appropriate also because it does not require us to keep track of which the miner is and which blocks they have mined so far[1] which may contain some uncertainty, especially because there is no specific field defined by the protocol referring to the miner.

In this case, all votes will be equally weighed ($weight = 1$) for each miner, with the more active miners inserting more votes in the block chain, since they have mined more blocks. To avoid cases where previously active and currently inactive miners influence the future decisions, we will keep a sliding window, i.e., we will only keep the last N blocks (votes).

5.4 Vote Location

One of the principal goals of this project is to find an appropriate way to insert the vote of each participant into the block chain. The challenge stems mainly from the fact that the blocks have an already defined and fixed structure with limited extensibility. By taking a closer look at the fields of the block header

[1] However, finding out the identity of the miner and associating it with a mined block is feasible in two ways:

- as the payout address hash of the reward in the coinbase transaction
- as a plaintext sometimes referred in the output script as "`mined by X`".

and the transactions, we can recognize several potential solutions that will have to be evaluated against the suitable criteria in the following option assessment.

Eventually, the users (SHs and HTUs) will use the *transaction locktime* to insert their vote, while as far as the miners are concerned, we will use the votes already injected by them in the *coinbase-transaction input script* (BIP 100) and the *header version* (BIP 101).

5.4.1 Evaluation Criteria

In order to decide over the best alternative for the insertion point of a vote, different alternatives had to be explored and evaluated, mainly against the following criteria:

- data overhead
- interference/incompatibility with functionalities used currently or in the future
- ambiguity of the vote, that is, the possibility to tell whether somebody abstained intentionally or just ignored the field
- packet clean format, i.e. the field values correspond to the field descriptions defined by the protocol

5.4.2 Option Assessment

We will investigate some methods to do soft-forking changes in order to give the possibility to all interested parties to include information into the blockchain. As already mentioned, there are two different positions where the votes should be appended, depending on the category by which the vote is cast.

More specifically, the **stakeholders** and **high-traffic users** are the ones that create and broadcast transactions, which means that they can only have access and alter the contents of their transaction and henceforth the vote has to be included *within the transaction*. Thus we recognize the following alternatives:

- **Locktime:** As explained, for values greater than or equal to 500,000,000 this field refers to the UNIX timestamp at which the transaction will be unlocked. Of course, if this value refers to some point in the past, the transaction will not be locked at all. So, for instance, values between 500,000,000 and 800,000,000 refer to dates between 05.11.1985 and 09.05.1995 that of course have no reasonable meaning or purpose. Hence we could take advantage of these values and append information regarding the votes. However this has as a consequence that we cannot use the functionality of the field, that is, lock a transaction until some later point in time.

- `OP_RETURN`: The script command (can be in both the `scriptSig` of an input or the `scriptPubKey` of an output) marks a transaction as provably unspendable/prunable output, which means that the script is marked as invalid and nobody can spend that output. This allows adding data to a transaction without the data ever appearing in the UTXO set. Of course this brings an extra overhead, since we have to include an extra (unused) output in the transaction [1].
- **Input sequence number:** This field is currently not in use and hence its value is always set to `0x FF FF FF FF`. This means that a transaction may not be considered to be final and hence be ignored by some nodes when it is not set to the maximum value.

The **miners** are not allowed to modify the transaction data, but only the block header fields and the coinbase transaction. Therefore their votes should be appended in either of these fields.

- **Coinbase input:** Since the input script of the coinbase doesn't have to be a valid script, any data can be appended in it, as is the case for the genesis block.[2] The BIP 100 proposal is based on this opportunity.
- **Version header:** The version number is already used for the BIP 101 proposal.
- `OP_RETURN`: The command can be appended in exactly the same way as described before, this time in the coinbase transaction.

Summarizing all available alternatives discussed above, we conclude to the comparative table 5.1.

5.5 Vote Format

In order to be able to recognize the votes in the chain, we will have to encode the information properly. For the three vote types, we will have the following formats:

- **User vote:** In order to be sure that the lock time value will not lock the transaction, we should pick a value between 500,000,000 (so that its a UNIX timestamp) that refers to 05.11.1985 and any UNIX timestamp in the past, say 01.01.2000, which is equivalent to 946,684,800.

[2]The coinbase of the genesis block contained the dated title of an FT article: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"

Locus	Field	Advantages	Disadvantages
Transaction	`Locktime`	No data overhead.	Possible conflicts with other proposals that may be using it – without a widely adopted protocol.
	`OP_RETURN`	No interference with any functionality. "Cleaner" packets and uniform application for transaction and blocks. Can be combined with other mechanisms that try to include information.	Data overhead (unused outputs in the transaction).
	Input sequence number	No data overhead.	May be used in the future (incompatibilities/interferences with replacement functionality).
Block	Coinbase input	Already implemented (BIP100). No interference with any functionality.	Data overhead.
	Version number	Already implemented (BIP101). No data overhead.	Ambiguous.
	`OP_RETURN`	No interferences. "Cleaner packets". Uniform application for transaction and blocks.	Data overhead.

Table 5.1: Comparative table of the different vote-locus alternatives.

Hence, in order to distinguish the votes for different proposals, as well as positive/negative votes from abstention, we can use two bits that are zero in the binary representation of 500,000,000:

0001 1101 1100 1101 0110 0101 0000 0000

Taking for instance the 29th and 30th bits, the former is used to indicate that we consciously vote and not abstaining, while the latter represents our negative or positive vote, respectively for bit values 0 or 1.

This scheme gives at the same time the possibility to use further bit pairs in the same fashion in order to vote for other proposals in parallel.

- **BIP 100:** According to this scheme, the miners are allowed to cast their vote by encoding 'BV' + BlockSizeRequestValue into coinbase `scriptSig`, e.g. "/BV8000000/" to vote for 8MB [5]. However, many miners do not conform to this scheme and just include blocks with the string "BIP100" in their coinbase [8].
- **BIP 101:** The version number has the 1st, 2nd and 30th bits set (`0x20000007` in hex) [7].

In all three cases, in order to be able to evaluate the data in a uniform manner, we will consider a binary vote format, that is, positive vote or negative vote (including abstentions).

5.6 Capturing the Data

When it comes to capturing and recording the data, we discern as before the three user categories:

- For the **high-traffic users**, we need to consider the data within a specific time context, say a month, year etc. For this reason, only the votes/transactions within a window of a certain size (in blocks) will be taken account of for the vote evaluation and the result computation.
- For the **stakeholders**, we only need a snapshot at a point in time. With this snapshot we can check how many unspent outputs they have and adjust the weight of the transaction vote by checking the next blocks in the chain without having further context.
- For the **miners**, a snapshot is enough as well, since we will just use the unity weight.

5.7 Platform

5.7.1 Emulation and deployment networks

For the needs of testing and measuring we will have to emulate the behavior of the three node categories and/or record their vote-casting activity. We will distinguish the following cases once again:

- For the **miners**, there is already the existing **mainnet**, where we can directly read out the votes and record the statistics.
- For the **high-traffic users** and the **stakeholders** we will have to inject the votes into the packets for each of the categories and hence the emulation will have to take place in the *regtest* network.

  For situations where interaction with random peers and blocks is unnecessary or unwanted, Bitcoin Core's regression test mode (regtest mode) allows instantly create a brand-new private block chain with the same basic rules as *testnet* (satoshis with no real-world value, more relaxed transaction checks), but one major difference: we choose when to create new blocks, so we have complete control over the environment [2].

  The reason we preferred the regtest network over testnet, is the fact that it runs locally and is hence easier to set up. Moreover, it allows backdating, that is, creating an arbitrary history by block mining in relatively short time, so that we can test the new blocks and transactions in our system in relatively short periods.

5.7.2 Development

Tools

For the actual development of the application and the realization of the programming logic, we will make use of the following programming tools/interfaces.

- **BitcoinJ:** An open source Java implementation of the Bitcoin protocol helping the development of Java (and not only) applications that interact with the Bitcoin network [9]. It can connect to the network, discover the network peers, download and store the blocks of the chainmaintain a wallet, send/receive transactions without needing a local copy of Bitcoin Core [10], provide listeners for events like a new block arrival etc.
- `bitcoind`**:** A program providing a full peer which we can interact with through RPCs.
- `bitcoin-cli`**:** A program that allows us to send RPC commands to `bitcoind` from the command line. Although the application will be written in Java and based on the BitcoinJ library, we will use an additional utility `BitcoinJSONRPCClient` that provides a programmatic interface with `bitcoin-cli` and allows access to the full functionality through RPCs.

Application structure

The application runs on two main threads:

- `TransactionGenerator`**:** This thread emulates the behavior of the users creating and announcing transactions to their peers in the network, including their vote regarding the block size. For the sake of emulation, there are two different user profiles being used: one for stakeholders, that is, users transacting with bigger amounts of bitcoins, and one for high-traffic nodes, i.e. users submitting numerous transactions of lower value.
- `VoteCounter`**:** At the beginning of the runtime, this thread goes through the whole blockchain and parses each past block for any possibly existing votes (of any category) in it. In parallel, there is a listener registered in this thread that parses all new blocks when a block-downloaded event is triggered. All parsed votes are stored in a local database.

The two threads explained above are called by the `EmulationStarter` class, once it has initialized the user wallet. Then the same class can call the appropriate class of the `VoteEvaluation` class in order to read the data from the database, evaluate the votes and extract the final results so far.

VOTES TABLE		
id	int	Auto-incrementing index.
block_hash	String	The hash of the block where the vote has been inserted into.
block_height	int	The block height for quicker reference on result evaluation.
transaction_hash	String	The hash of the transaction where a user inserted the vote or of the coinbase transaction in the case of the miners.
vote_type	Enum	The vote type (BIP100, BIP101, USER), so that it can later be parsed appropriately.
vote_value	boolean	TRUE for positive votes, FALSE for negative votes/abstentions.
vote_weight	float	The sum of the unspent outputs of the transaction for the user votes (that will be used for the SH-view evaluation. Unused for the rest.

Table 5.2: Structure of the VOTES table.

Two further classes Utils and StatsKeeper are used for general-purpose static methods and storing statistics into a Matlab file.

5.7.3 Storing the data

The database scheme we are going to use will consist of only one table (Table 5.2) that describes the votes in a uniform manner for the high-traffic users, stakeholders and miners, although some fields may not apply to both cases, e.g. vote_weight. What will eventually differ is the way we parse the blocks / transactions to extract these data, as well as the weight we will attribute to them when evaluating the vote results.

Chapter 6

Evaluation

In this chapter, we will present some interesting results regarding our voting system for the users, as well as the already existing proposals BIP 100 and BIP 101 for the miners.

As far as the users are concerned, we have already examined two possible user views: the stakeholders and the high-traffic users. Each user, depending on their behavior, participates in the system and hence in the voting procedure more as a SH and less as a HTU or vice versa.

Although all users vote in a uniform way, without being classified explicitly in one of the two views, the voting procedure should be able to let both views express their preference (vote) distinguishing them from one another. An important parameter to be assessed in our system should thus be the extent to which the result extracted for each view really corresponds to the result that each view voted for.

To exemplify this point, we suppose that the users are called to vote for or against a given proposal. If the users whose behavior fits more to the description of the SH-view decide to vote with a certain ratio A in favor of the proposal, whereas the users better described by the HTU-view decide to vote with a ratio B in favor of the proposal, then the final outcome of the vote evaluation should depict this situation by outputting a result close to A for the SH-view and B for the HTU-view.

As for the miner voting system, results that could be of interest and will be presented are the following:

1. The delay of the block-parsing procedure for the blocks of the main network.

2. The participation of users as far as the proposals BIP 100 and BIP 101 are concerned.

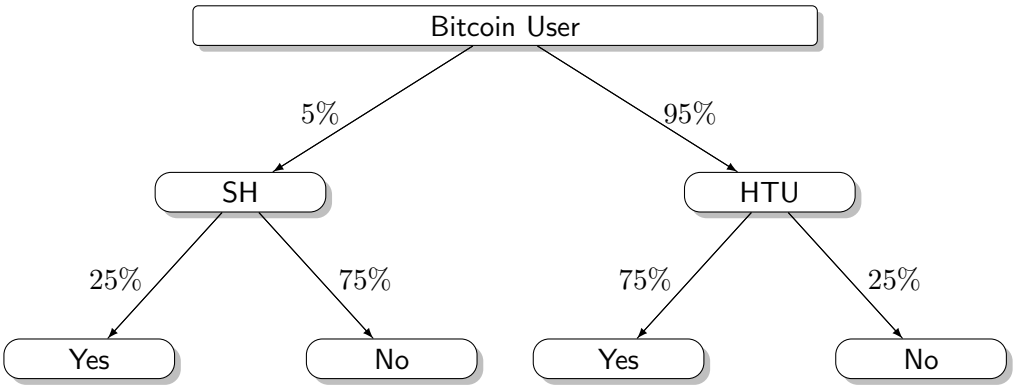

Figure 6.1: Emulation scenario for users

6.1 User Evaluation Scenarios

In order to test the expressiveness/validity of the metrics we chose for the users (SHs and HTUs), we will consider the scenario of Figure 6.1. As can be seen in the graph, our scenario includes the two different user views, SH and HTU, and therefore we will consider users that adopt one of the two distinct behaviors, either closer to one or the other view.

We define the following parameters for our emulation:

- **SH/HTU frequency:** The SHs and the HTUs generate 5% and 95% of the total transactions, respectively.
- **YES/NO ratio:** Moreover, the former decide to vote in favor of the proposal (positive) with a ratio of 25%, leaving the rest 75% cast a negative vote, while as far as the latter view is concerned, 75% vote for and 25% against it.
- **held value:** Also, we set the total bitcoins held by the SHs to be 100 times more than the overall value held by the HTUs, so as to make the two behaviors more distinct from each other.
- **evaluation window:** In order to discard stale votes, that is, votes coming from users that may have been active in the past but not anymore, we need to define a relatively short evaluation window. In our case, this is set to 50 blocks.

Taking the above into consideration, apart from the standard scenario, we will also try different values for the aforementioned parameters (Table 6.1), in order to examine their influence in our system's performance.

Scenario	(f_{SH}, f_{HTU})	$\left((\frac{YES}{NO})_{SH}, (\frac{YES}{NO})_{HTU}\right)$	$Value_{SH}/Value_{HTU}$	HTU Window
Standard	$(5\%, 95\%)$	$(25\%, 75\%)$	100	50
SH/HTU frequency	x	$(25\%, 75\%)$	100	50
YES/NO ratio	$(5\%, 95\%)$	x	100	50
Held value	$(5\%, 95\%)$	$(25\%, 75\%)$	x	50
Evaluation window	$(5\%, 95\%)$	$(25\%, 75\%)$	100	x

Table 6.1: User evaluation scenarios.

6.2 User Evaluation Results

Standard scenario. In Figure 6.2 we can see the result of the vote parsing and evaluation (i.e., multiplication by the appropriate factor and summing up) while more blocks are being mined and added to the block chain. The dashed line corresponds to the theoretical values we set in our scenario, the dots to the actual measurements we extracted and the solid one to the mean value of our measurements.

From the graph we can point out that the two user views give accurate results representing the actual voting results of the SHs and the HTUs accordingly. Not only is the mean value of the positive-vote percentage for each view close to what they voted (less than 5% in both cases), but also the deviation of each measurement is very small from this mean value.

The reason for the discrepancy of the SH-view result is that the HTU users still hold some bitcoin value and vote with a considerably higher percentage positively. On the other hand, as far as the HTU results are concerned, we observe that the voting results are lower than the original votes of the HTU users, because the votes of the SHs has still been considered.

For the following scenarios, we will only use the mean value for each view for each setting, i.e., set of scenario parameters, in order to extract some conclusions regarding the behavior of the scenario parameters.

Different SH/HTU frequencies. In the graph of Figure 6.3, the mean results for the positive votes of each view are plotted as a function of the percentage of the SH votes over the total votes cast.

As expected, when there are no SHs voting, the votes of the HTUs will determine the result of the SH view as well, in spite of the low overall held value. On the other hand, as the proportion of SH votes increases, the result for the HTU view decreases rapidly towards the theoretical value for the SH view, since the votes cast by the latter make up for a bigger part of the total votes.

As a consequence, when 30% of the votes come from SHs, the result for the HTU view is 15% below the expected value. On the contrary, the corresponding result for the SH view is less than 4% away from the theoretical value, an expected fluctuation due to the randomness inserted into the vote generation.

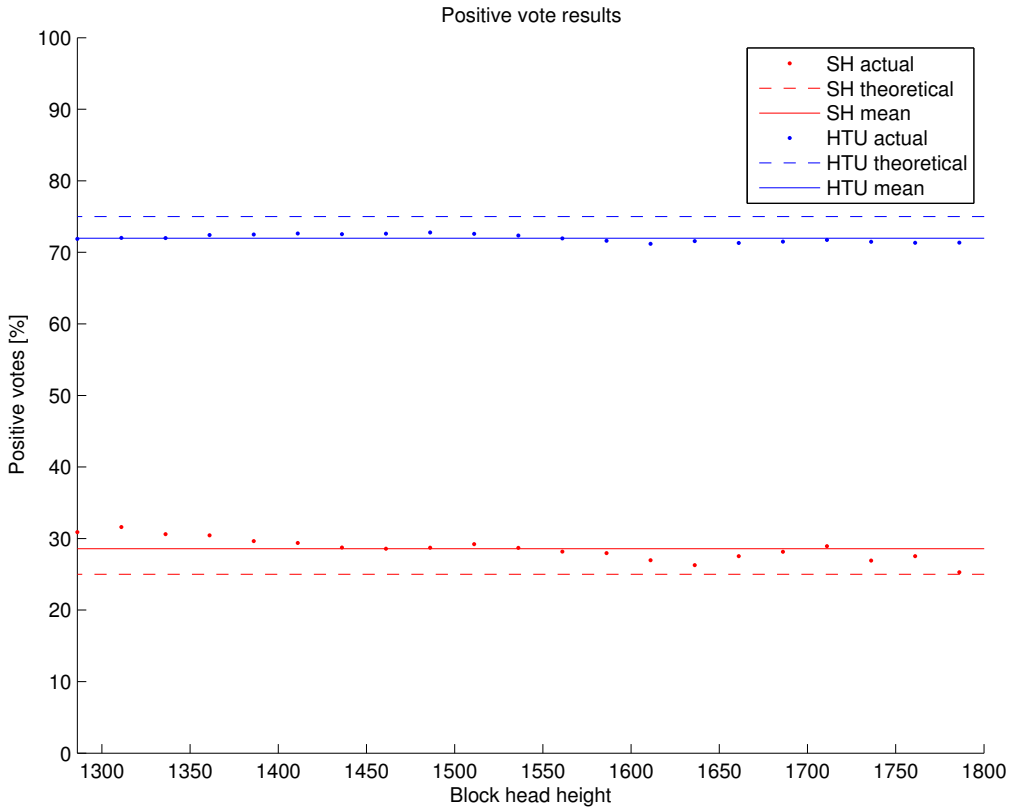

Figure 6.2: User positive-vote results for the SH and HTU views for the standard scenario.

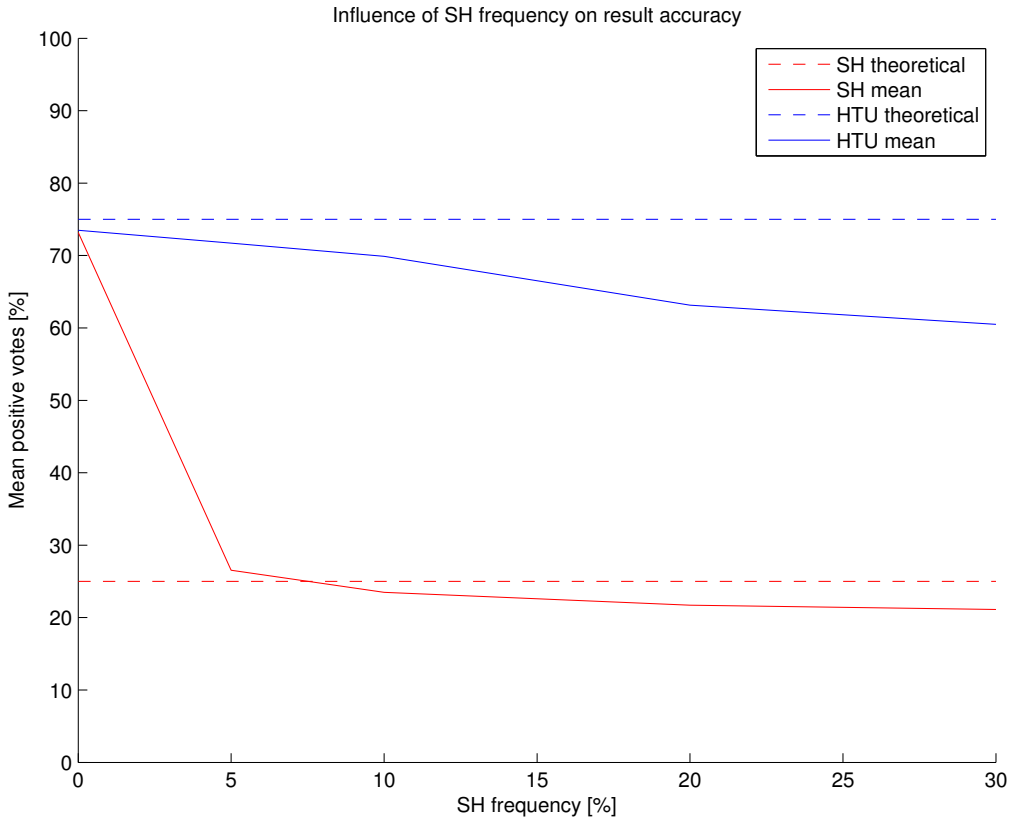

Figure 6.3: User mean positive-vote results for the SH and HTU views, but different division of users between the SH and HTU views.

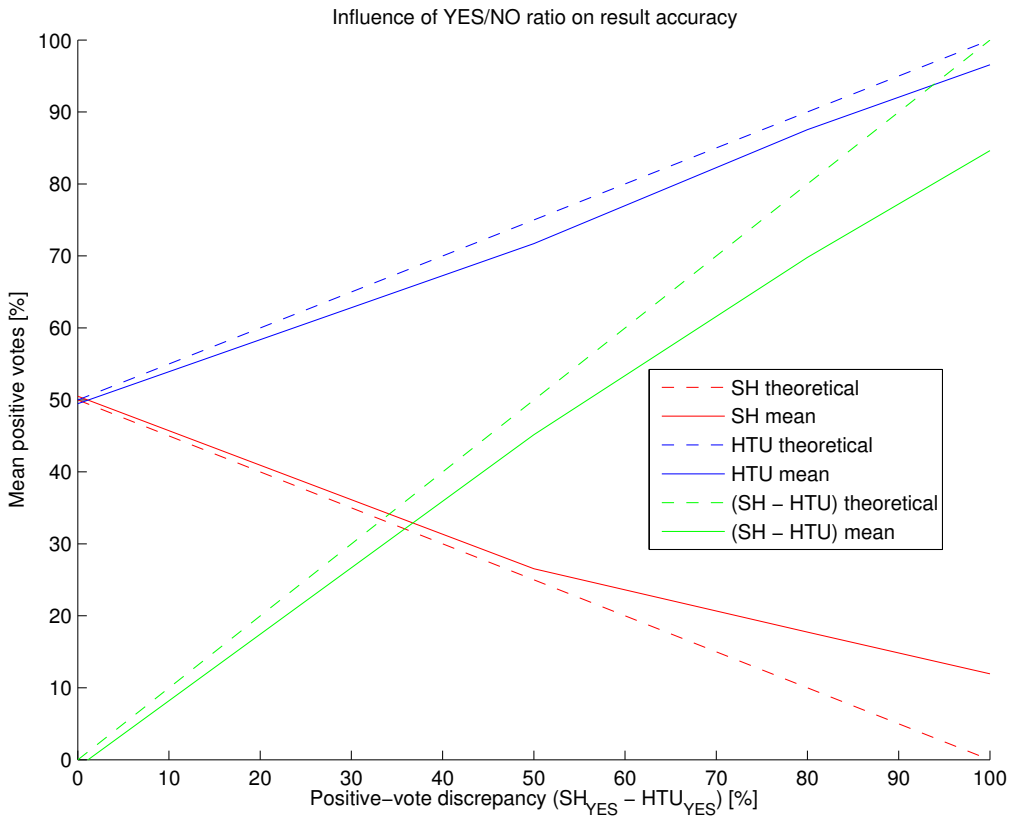

Figure 6.4: User mean positive-vote results for the SH and HTU views, but different discrepancies of YES/NO ratio.

Different YES/NO ratios. In each experiment of this scenario, the SHs voted positively at a percentage of $x\%$, while the HTUs at a percentage of $100\% - x\%$, thus forming a difference of $100\% - 2x\%$ between them. In Figure 6.4, the vote results are plotted as a function of the difference between the YES-votes of the SHs and HTUs. To exemplify, when this difference is 0%, both voted with a percentage of 50%. On the other hand, when the difference is 100%, none of the SHs and all of the HTUs voted positively.

As experimentally confirmed, when both of the views voted had the same voting behavior, i.e., 50%, none of them influenced the results of the other, since they voted unanimously. However, when they presented exactly opposite behaviors, they influenced the other view the most, hence affecting the accuracy of the overall results.

Different held values. In this scenario, the influence of the ratio of the held value of the SHs over that of the HTUs is examined.

As is to be observed in the semi-logarithimc diagram of Figure 6.5, the SH

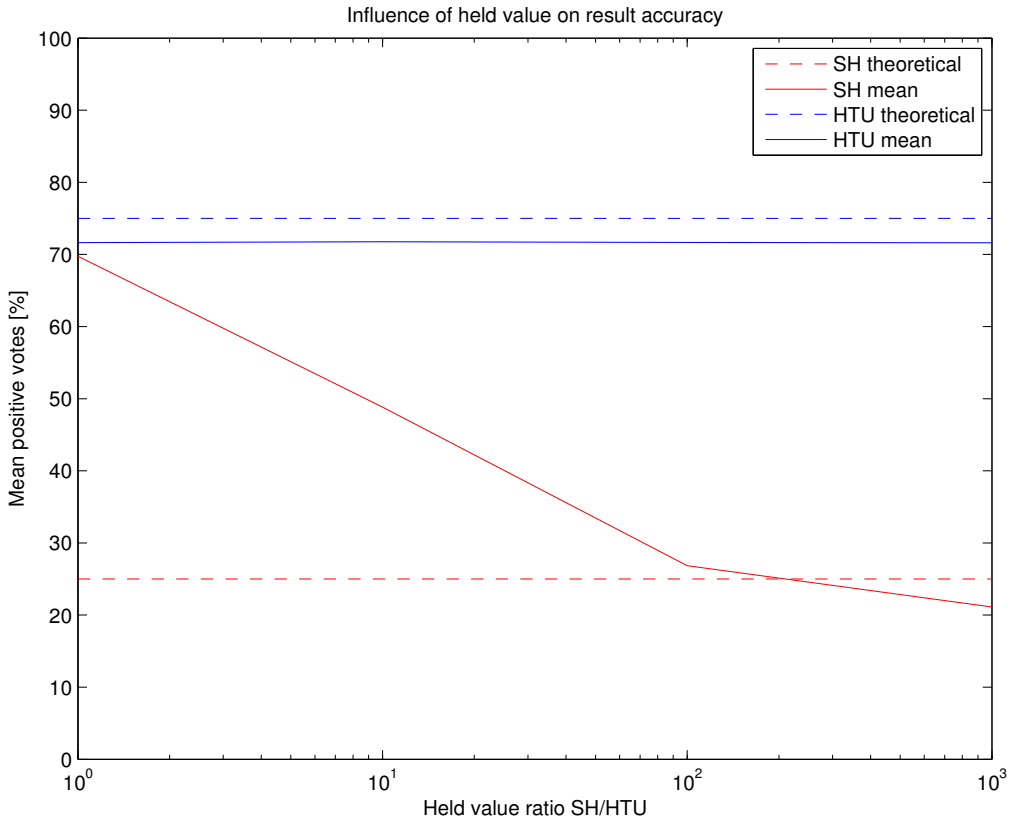

Figure 6.5: User mean positive-vote results for the SH and HTU views, but different amounts of held bitcoin value ratio.

view is almost entirely determined by the HTUs when each user hold the same total bitcoin value. The reason is that the HTUs are much more in number, hence they hold more value in total and can impose their voting behavior over that of the SHs. As this ratio increases above 100, the SHs can determine the result of their view relatively unaffected by the votes cast by the HTUs. Nonetheless, the results of the HTU view remain unaltered by the variations of the total held value, since it does not come into its computation.

Different window sizes. As illustrated in Figure 6.6, the variation in the value of the evaluation window left the SH view unaffected (since it doesn't come into its calculation), while the HTU view is only lightly influenced by the window size.

This happens because in our simulation scenario, the users had a similar behavior throughout the emulation period, while the window mechanism is only used to make the results immune of stale votes, that is, votes cast by users in the past, when their voting behavior was different from the behavior of the users that have voted recently.

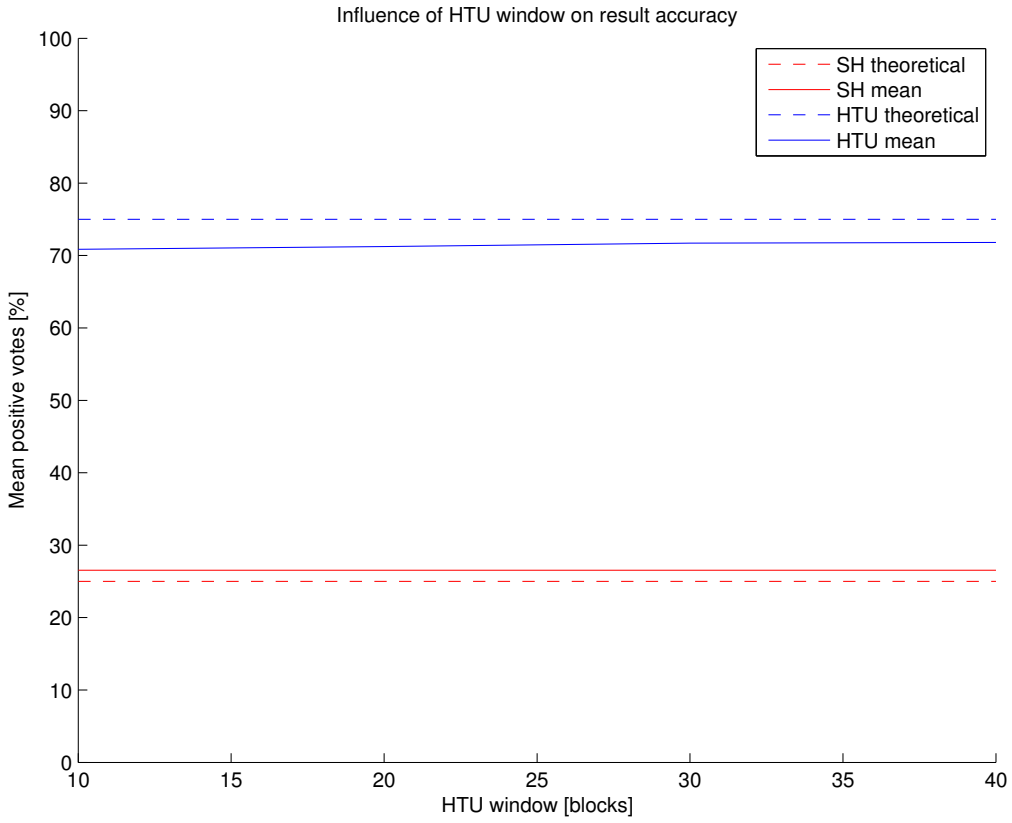

Figure 6.6: User mean positive-vote results for the SH and HTU views, but different evaluation windows.

Summary. Although sending out a big amount of transactions could have a strong impact on the results for the HTU view, a user cannot influence the extracted voting result arbitrarily. This means that we can neither insert as many votes as we want, nor insert only our votes excluding those of other users, because including transactions comes with a cost, and more precisely the transaction fees that have to be paid in order to include our transaction into the block. The more users that want to insert transactions in the blocks, the higher the contention and hence the minimal fee that has to be paid for the transaction (i.e. the vote) to be included in the chain.

As proven by means of emulation, although a SH and a HTU vote in the same uniform way, our vote-counting system and evaluation technique successfully distinguished the preferences of the two groups with good precision.

Nonetheless, this categorization of a user might not always be very clear, thus affecting the accuracy of the results. Among the factors that can influence the distinctness of the two views are the following:

- **held value:** The bigger the difference between the amount of bitcoins held by the SHs and HTUs, the easier to extract correct results for the SH view.
- **SH/HTU ratio:** For percentages close to 50%, i.e., equal proportion of SHs and HTUs, we expect the HTU view to present results more biased, therefore inaccurate, towards the SHs.

At the same time, the evaluation seems to be to a high degree immune with respect to the parameters below:

- **evaluation window:** As long as the minimal sample is collected, the results will have the desired accuracy. Bigger sample sizes on the other hand may give more importance to currently inactive users.
- **YES/NO ratio:** As seen, both SHs and HTUs voting positively with exactly the same ratio returns zero discrepancies for the results of the two views. However, even different YES/NO ratios for the SHs and the HTUs influence only up to a limited extent the results for the other view.

6.3 Miner Further Results

As far as the BIP100 and BIP101 proposals are concerned, it would be interesting to observe their adoption rates, as well as their evolution through time. From the graph of Figure 6.7 the following points are to be mentioned:

- The BIP100 proposal was much more widely adopted than BIP101, since the votes for the former can be found in about 10 times more blocks than those for the latter.

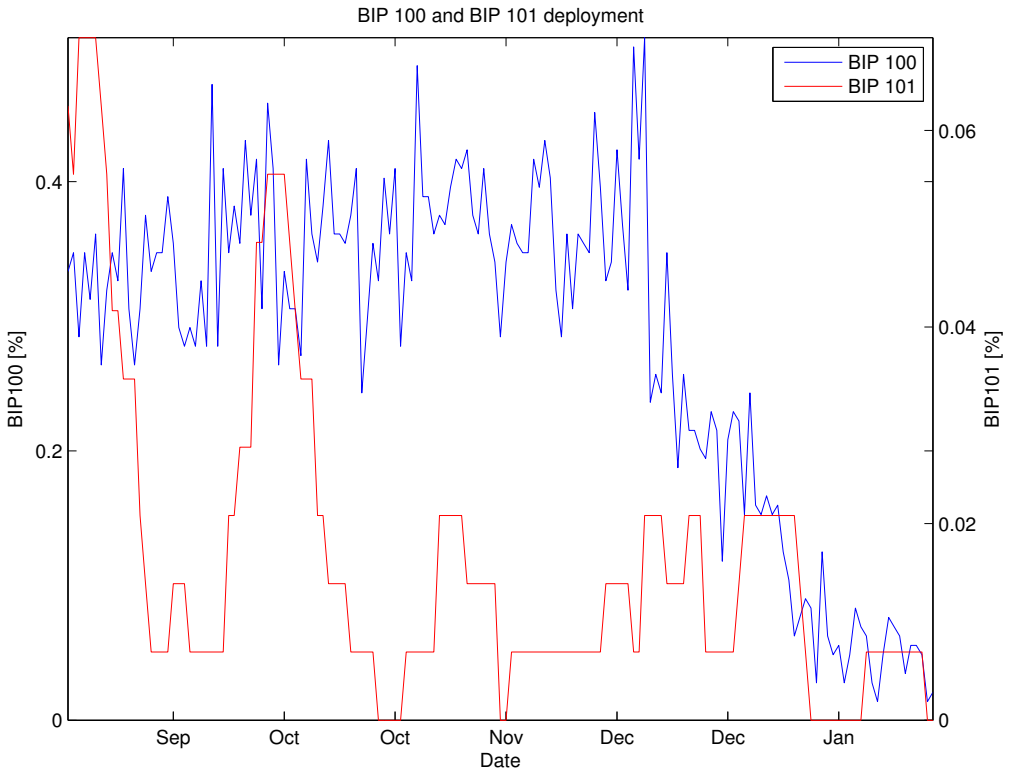

Figure 6.7: Deployment of BIP 100 and BIP 101 proposals

- Both proposals present a declining tendency with less and less blocks including a corresponding vote for any of the two.

Chapter 7

Conclusion

In the course of this project, we addressed the need of the Bitcoin ecosystem to converge towards a consensus with respect to an existing proposal.

Our starting point was the identification of the problem; then we recognized, analyzed and assessed the various options and developed a voting scheme according to the result of this evaluation. Afterwards, using the block-size debate as a study case, we emulated a Bitcoin network where users with different characteristics, as far as their financial activity is concerned, inserted their votes in the Bitcoin block chain. Eventually, we counted the votes and evaluated them using the appropriate metrics, demonstrating that the voting behavior of the users could indeed be reflected with satisfactory accuracy in the extracted results.

Bibliography

[1] : Bitcoin wiki. https://en.bitcoin.it/wiki/

[2] : Bitcoin.org, developer guide. https://bitcoin.org/

[3] : Cex.io, bitcoin fork explained. http://blog.cex.io/bitcoin-dictionary/what-is-bitcoin-fork-14622

[4] : Wikipedia. https://en.wikipedia.org/

[5] Garzik, J.: Bip 100 theory and discussion. https://github.com/jgarzik/bip100

[6] : What is the bitcoin block size debate and why does it matter? http://www.coindesk.com/what-is-the-bitcoin-block-size-debate-and-why-does-it-matter

[7] : Bitcoin improvement proposals. https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki

[8] : Blockchain blocks version. https://data.bitcoinity.org/bitcoin/block_size_votes/2y?c=block_size_votes&t=bar

[9] : The bitcoinj api. http://www.javaworld.com/article/2078482/java-web-development/bitcoin-for-beginners--part-3--the-bitcoinj-api.html

[10] : Bitcoinj. https://bitcoinj.github.io