Inputs Reduction for More Space in Bitcoin Blocks

Michal Zima
Faculty of Informatics
Masaryk University
Brno, Czech Republic
E-mail: xzima1@fi.muni.cz

***Abstract*—Bitcoin blockchain is growing big, yet its transaction capacity hit its limits in 2017. One solution heavily discussed within the bitcoin community is to let it grow faster by increasing size of the blocks. However, making transactions smaller instead of blocks bigger is more sustainable and also brings other additional benefits. In this paper, we focus on space wasted by non-cryptographic use of hashes and uncompressed numbers within transaction inputs. In our analyses, we show that transactions can be made approximately 16% smaller, depending on their complexity.**

I. Introduction

Scalability has been one of the hot topics in the bitcoin community in recent years. In a surge of Bitcoins popularity and spreading usage, the number of transactions rose to the point that transactions started competing with one another for being included in a block. This led to an emergence of a transaction fee market and a development of various scaling proposals.

All processed transactions come with a cost. They need to be stored by all full nodes indefinitely. Most scaling proposals focus on increasing transaction capacity by merely allowing more transactions to be processed. While this lessens the issue of low transaction capacity to some extent, it increases the cost of storage. Therefore, we focus on the issue of lowering this cost: allowing more transactions to be processed without increasing the storage requirement beyond their current growth.

The main contribution of this paper lies in the reduction of wasteful, redundant data in transaction inputs. We propose both fields of an output reference to be reduced: a transaction hash to its unique prefix and an output index to a variable integer. Our work may shrink transactions and therefore also blocks for approximately 11–16%.

The remainder of this paper is organised as follows. Section II presents previous work related to the problem. In Sect. III we discuss details of the structure of bitcoin transactions. The following Sect. IV presents our proposed solution for reducing the size of transactions with compatibility fallbacks for software not capable of full support of the proposal in Sect. V. Section VI discusses how the input reduction affects existing cryptocurrencies and required changes. Analyses are presented in Sect. VII. Section VIII provides final conclusions.

II. Related Work

We divide Bitcoin scaling proposals for scaling its blockchains transaction capacity into three classes. We call as *blockchain-level scaling*, *block-level scaling* and *transaction-level scaling*. However, blockchain-level scaling is far beyond the scope of this paper; therefore we omit details of this class. Nonetheless, a good overview of it is given in [1].

A. Block-level Scaling

This class of proposals focuses on methods to make blocks bigger so that more transactions can fit inside. A wide range of formal proposals was developed.

Garzik et al. devised a scheme to determine the block size limit from miners votes [2]. A similar concept was developed by BtcDrak [3]. Blocks that contain votes for a block size limit increase are more difficult to mine, therefore imposing an implicit cost for the voting miners.

Andresen proposed a gradual growth of the block size limit from 1MB limit to 8MB—at a given rate in 20 years time [4]. Garzik simplified this proposal into a simple one-time increase of the block size limit to 2MB [5]. Andresen later followed this work with a further refinement of related parameters [6].

Wuille suggested a method of loosely following a technological growth, namely the growth internet bandwidth, which translates into a 4.4% increase of the block size limit every ∼97 days (i.e., ∼17.7% per year) [7]. A different proposal by t.khan aims for automatic adjustment of the block size limit in a way similar to mining difficulty [8]. Its goal is to keep blocks on average 75% full, wherefore the limit is increased or decreased accordingly every 2016 blocks, based on sizes of blocks from the previous period. Another dynamic adjustment scheme was proposed by Chakraborty: instead of taking into account only the actual size of recent blocks, it also calculates with overall transaction fees spent in those blocks [9].

A combination of static and dynamic increasing of the block size limit can be found in work of Sanchez [10]. They proposed first increasing the limit in several steps to given values, subsequently followed by a dynamic mechanism for increases on as-needed basis.

A novel approach to enlarging space in blocks is represented by extension blocks [11]. Instead of changing the legacy block size limit, authors create an extension to every block. This extension contains a distinct transaction history, i.e., transactions within extension blocks are knowingly carried out within the extension blocks. Nevertheless, the money supply is shared with standard blocks and transaction history, while there is firm protection against double spending. This proposal

is related to an earlier work on "sidechains", which build not just extensions to blocks, but a separate blockchain [12].

However, the common issue of these competing proposals was not being widely accepted within the bitcoin community as the "right" scaling solution.

B. Transaction-level Scaling

In contrast to block-level scaling proposals, there is a significantly lower focus on finding inner reserves for scaling within blocks, i.e., in transactions.

The most significant work in this area is a proposal call Segregated Witness [13]. Under this proposal, input part of the script (called a "witness") is moved to a separate structure—a witness tree that is attached to the block, therefore not affecting the original block size as it is perceived by legacy nodes. This proposal has already been implemented into Bitcoin—in a way that is backward compatible with legacy software.

A competing proposal called Flexible Transactions aimed at reworking the transaction structure (which is detailed in the following Sect. III) so that it would be possible to omit fields that are not used [14]. The author claims that this improvement would make transactions 3% smaller on average.

III. Transaction Structure

Every bitcoin transaction consists of a version field, a set of inputs, a set of outputs and a time lock field. On a binary level, the version and time lock fields are 32-bit integers and both sets are preceded by a variable-length integer that efficiently encodes a number of items in each set. [15]

Every input comprises a reference to some previous output—in the form of outputs transaction hash and a 32-bit integer index of the output within its transaction—, an input part of a transaction script and a 32-bit integer sequence number. The script is a byte array, again preceded by a variable integer with its length.

Every output comprises an amount field and an output part of a transaction script. The script is represented in the same way as in inputs, including its length. The amount field is a 64-bit integer.

Clearly, the structure of bitcoin transactions is already very dense. It would be needed to remove fields (and thence also their related functionality) to make transactions smaller. Among the very few other options is moving the script out of inputs, which is being done by the Segregated Witness technique [13].

Another option is discussed in this paper: a focus on output reference in inputs.

IV. Inputs Reduction

Output reference in inputs is formed by two fields of a fixed length: a 32-byte transaction hash and a 4-byte index. While the transaction hash plays a cryptographically significant role within the blockchain when storing transactions in blocks (it prevents transaction replacement attack on mined blocks), its role within inputs is different. In an input, the hash provides a unique identifier of the transaction where the referenced output is situated. The index then specifies what number the output is within the said transaction.

Given the vast value space of 2^{256} (the transaction hash) and a very limited number of transactions that take place within the bitcoin system, we can say that a probability of identifier collision remains sufficiently negligible even for significant reductions to 160 or 128bits.

Nonetheless, we propose to use as few bytes as possible—through the use of a variable integer-like mechanism—and compensate for collisions that *will* occur.

A. Collision Resolution

At the moment of transaction verification, there might be several transactions whose hashes share the same prefix as is specified in the input being verified. To resolve this collision, the verifier takes into account only transactions that were included in blocks of the same or lower height than the transaction being verified. Also, to limit the likelihood of a collision, only unspent outputs can match to the given input.

It needs to be noted that collisions beyond the stated level are not allowed. This way we ensure that referenced outputs can be unambiguously identified at all times. Therefore, computational requirements are not unnecessarily increased by iterating through a set of candidate outputs and verifying their scripts.

B. Output Index Size Reduction

Output index is the second component of an output reference in inputs. As we show in Sect. VII, 99.97% of bitcoin transactions contain only up to 252 outputs. We can leverage this finding to further reduce the size of inputs.

Instead of a fixed-size 4-byte field, we propose usage of a variable integer. A variable integer is a common data structure in Bitcoin and allows to store values 0–252 in a single byte—larger values then need 3 or 5 bytes. This means that almost all outputs could be referenced with a 1-byte index field.

On the other hand, all coinbase transactions would need 5 bytes instead of 4 to encode the value `0xFFFFFFFF` that is used as an index in their dummy input. However, this overhead is negligible in the total impact.

V. Fallbacks

In situations in which a wallet software is unable to determine a unique reduced transaction hash, e.g., it may be still synchronising with the network, constructs and signs the transaction offline or simply lacks this capability, it can still create a transaction. There are generally two possible approaches:

1) "Try-and-see"—use the best possible estimate. If such transaction is rejected by peers, the hash length can be extended until the transaction gets accepted.
2) Not to use reduced hashes—i.e., to reference the transaction with its full transaction hash, which is guaranteed to be unique [16].

Neither of these solutions is ideal: the first one gives out privacy to the peers and makes the node susceptible from an

attack, in which other nodes may force it to use a full hash; the second one does not suffer from this issue but is even slightly more inefficient than current bitcoin transactions.

VI. Compatibility with Current Cryptocurrencies

Our proposal of hash and index reduction is a change not directly compatible with todays cryptocurrencies[1]. However, it can be implemented in the form of a soft fork extension. This means that while all mining nodes need to upgrade and support this change, other nodes may work within the network even without upgrading.

The reduced structures constitute a new version of a bitcoin transaction. Legacy implementations will, therefore, ignore them as non-standard. An upgrade is necessary not only in order to spend from the new transactions, but also to support their broadcast through the network.

Overall necessary changes are twofold: in software and in the bitcoin protocol.

A. Software Changes

From the perspective of software internals, a new structure for determining a unique prefix might be needed. For efficiency of finding a unique reduced hash, we suggest usage of a trie-based index. Software that already uses a trie-like structure for indexing the UTXO set will probably not need significant changes.

B. Protocol Changes

SPV clients do not store the UTXO set and therefore are not capable of determining an optimal length of reduced hashes. They can either employ a fallback strategy as discussed in Sect. V or rely on the network to provide them necessary data.

We see multiple possible approaches to implement this support. Nonetheless, there is no clear best option; therefore we outline all of them and leave the final choice on developers.

1) On Demand Prefix Calculation: The most straightforward way of providing SPV nodes with a unique prefix is to simply offload the task of finding it to its neighbouring full nodes. An SPV node directly asks for a unique prefix of a give transaction output. This approach is the least communicational and computational intensive, but it is also the one with the worst privacy properties—the full node is able to link the output in question to the node with high probability.

2) Prefix Resolving: Another possibility for an SPV node is to guess a prefix and ask some of its neighbouring full nodes to return all matching transaction outputs. This approach may be communicationally expensive if a larger number of transaction outputs match the prefix (although a node should be able to refuse to return them if their count is higher than some threshold). Nonetheless, the SPV node can compute the unique prefix by itself from returned data.

On the other hand, if the prefix is longer than necessary, there will be only a single match. While this gives a usable result, this prefix may not be optimal (i.e., the shortest possible). The node might want to repeat the query for a shorter prefix. Also, this scenario is privacy-wise equivalent to the previous approach.

3) Prefix Quantification: A similar approach to the prefix resolving is to not actually give the asking node the full list of transaction outputs. Instead, the full node returns a number of matching outputs. This approach saves most of the traffic between the nodes, although it leaves the SPV node with information "usable"/"not usable." Therefore, it might require more communication rounds for the SPV node to obtain intended information.

4) No Change: Even without special protocol messages, there is still a way to support hash reduction for SPV nodes as indicated in Sect. V. A node constructs a transaction using a short transaction output prefixes and broadcasts it. If the chosen prefix is ambiguous, the neighbouring node rejects the transaction together with an indication which output prefix was not unique, thus giving the originating node a chance to construct a new transaction and try again.

Apparently, this induces more traffic between the nodes and similarly to the other approaches may leak a connection between the outputs used and the SPV node.

VII. Analyses

We evaluate potential impact of our proposals on Bitcoin. We choose Bitcoin, because it has the largest transaction base among cryptocurrencies. We show total distribution of the number of outputs within transactions in order to illustrate the relevance of output index reduction. We then analyse the UTXO set to obtain a set of unique reduced hashes and analyse their lengths.

A. Output Indices

The goal of analysing output indices is to see how big indices occur in bitcoin transactions (i.e., how many outputs transactions usually have). That helps us estimate the impact of output index compression.

We analysed the UTXO set. Out of 54,975,282 transaction outputs, 82.7% were of index less or equal to 252. These could be referenced from inputs with only a single byte. The rest indices could be encoded in inputs with 3-byte variable integers. Compression of output indices within inputs spending the whole UTXO set would, therefore, save 139 MiB of space. Given a transaction with two inputs and two standard outputs (size of which is typically 373B), we save 6 bytes, i.e., 1.6%.

Another interesting insight can be derived from historical data. Out of 298,454,759 transactions we analysed, 99.97% transactions contained up to 252 outputs. This indicates that smaller transactions are more common than the UTXO set would suggest. The distribution of the number of outputs in transactions is visualised in Fig. 1.

B. Output Hash Prefixes

Average output transaction hash prefix is related to the number of unspent outputs in the whole set, because collisions

[1]We consider only cryptocurrencies derived from Bitcoin.

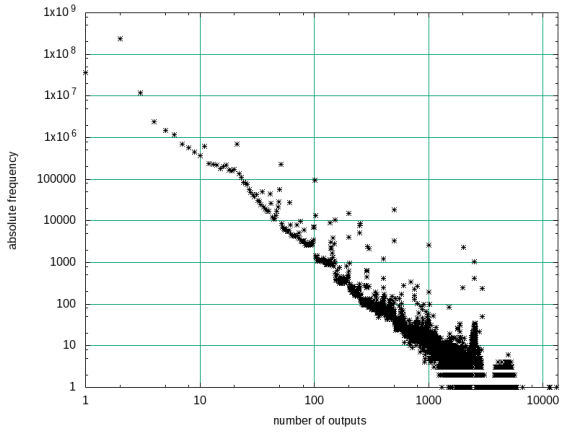

Fig. 1. Distribution of the number of outputs in transactions.

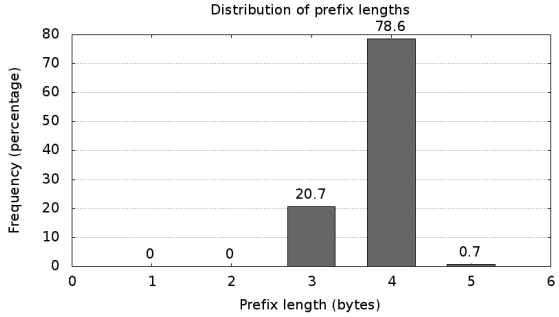

Fig. 2. Distribution of lengths of unique hash prefixes within UTXO.

are evaluated only within the current UTXO set. Distribution of prefix lengths is shown in Fig. 2. In the set of unspent outputs, most prefixes are unique within 4 bytes.

In the above-mentioned transaction this result translates into savings of 54 bytes, i.e., 14.5%. Together with compressed output index, we can save 16% in such transactions.

VIII. Conclusion

Scaling the bitcoin blockchain beyond its transaction capacity has been a subject of many efforts within the bitcoin community. However, most of the proposals focus only on methods for raising the block size limit to simply allow more transactions to be processed. That would have an impact on nodes that store and provide the bitcoin blockchain to the network—doing so would become more expensive.

In this paper, we addressed this issue from a different angle—by proposing a method to reduce the size of individual transactions, which facilitates the same goal, but without the aforementioned cost. Transaction inputs contain output references: by a transaction hash and output index. We reduce the size of the output index to 1B from 4B and the size of the transaction hash to 5B from 32B in most cases. By doing so, we reduce transactions approximately 11–16% in size.

For future work, we want to focus on optimising the implementation side to minimise the burden caused by the need for the nodes to be able to look up unique prefixes to transaction hashes.

References

[1] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gn Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," in *Financial Cryptography and Data Security*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, vol. 9604, pp. 106–125. [Online]. Available: http://link.springer.com/10.1007/978-3-662-53357-4_8

[2] J. Garzik, T. Harding, and D. V. Johannsson, "BIP 100: Dynamic maximum block size by miner vote," Jun. 2015. [Online]. Available: https://github.com/jgarzik/bip100/blob/master/bip-0100.mediawiki

[3] BtcDrak, "BIP 105: Consensus based block size retargeting algorithm," Aug. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0105.mediawiki

[4] G. Andresen, "BIP 101: Increase maximum block size," Jun. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki

[5] J. Garzik, "BIP 102: Block size increase to 2mb," Jun. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki

[6] G. Andresen, "BIP 109: Two million byte size limit with sigop and sighash limits," Jan. 2016. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0109.mediawiki

[7] P. Wuille, "BIP 103: Block size following technological growth," Jul. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0103.mediawiki

[8] t.khan, "BIP 104: 'Block75' – Max block size like difficulty," Jan. 2017. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0104.mediawiki

[9] U. Chakraborty, "BIP 106: Dynamically Controlled Bitcoin Block Size Max Cap," Aug. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0106.mediawiki

[10] W. Y. Sanchez, "BIP 107: Dynamic limit on the block size," Sep. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0107.mediawiki

[11] C. Jeffrey, J. Poon, F. Indutny, and S. Pair, "Extension Blocks," Mar. 2017. [Online]. Available: https://github.com/tothemoon-org/extension-blocks/blob/master/spec.md

[12] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timon, and Pieter Wuille, "Enabling Blockchain Innovations with Pegged Sidechains," Oct. 2014. [Online]. Available: https://blockstream.com/sidechains.pdf

[13] E. Lombrozo, J. Lau, and P. Wuille, "BIP 141: Segregated Witness," Dec. 2015. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

[14] T. Zander, "Flexible Transactions," Jul. 2016. [Online]. Available: https://zander.github.io/posts/flexible_transactions/

[15] A. M. Antonopoulos, *Mastering Bitcoin*, first edition ed. Sebastopol CA: O'Reilly, 2015.

[16] P. Wuille, "BIP 30: Duplicate transactions," Feb. 2012. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0030.mediawiki