

# Txilm: Lossy Block Compression with Salted Short Hashing

Donghui Ding<sup>1,2</sup>, Xin Jiang<sup>1,2</sup>, Jiaping Wang<sup>3,1</sup>, Hao Wang<sup>3</sup>, Xiaobing Zhang<sup>3</sup>, Yi Sun<sup>1</sup>

<sup>1</sup>Institute of Computing Technology, Chinese Academy of Sciences

<sup>2</sup>University of Chinese Academy of Sciences

<sup>3</sup>Monoxide Dev Team

## Abstract

Current blockchains are restricted by the low throughput. Aimed at this problem, we propose Txilm, a protocol that compresses the size of transaction presentation in each block to save the bandwidth of the network. In this protocol, a block carries short hashes of TXIDs instead of complete transactions. Combined with the sorted transactions based on TXIDs, Txilm realizes 80 times of data size reduction compared with the original blockchains. We also evaluate the probability of hash collisions, and provide methods of resolving such collisions. Finally, we design strategies to protect against potential attacks on Txilm.

## 1 Introduction

Blockchains have been applied to wide areas such as cryptocurrencies and finance. However, current blockchains are restricted by the low throughput. For example, Bitcoin only handles 7 transactions each second, while PayPal achieves 500 transactions/sec throughput and Visa even 4000 transactions/sec [6].

Aimed at the throughput problem of blockchains, various schemes are proposed including the compact blocks [3, 8] in Bitcoin. A compact block carries only 32-byte TXIDs instead of complete transactions (300–400 bytes roughly), and the network bandwidth consumed by each transaction is thus reduced by around 10 times. Compact blocks are safe because most transactions have already been stored in the memory pool of each Bitcoin node.

On the basis of compact blocks, we propose Txilm, which uses the short hash of TXIDs to represent the transactions in the blockchain. This further reduces the size of the transaction representation in each block to around 40 bits. Txilm is simple without using additional data structures such as bloom filters or IBLTs (invertible bloom lookup tables). Furthermore, our proposal doesn't rely on consistent memory pools across full nodes.

We also optimize Txilm by sorting transactions based on TXIDs. Such optimization reduces the transaction representa-

tion in a block to 32 bits, which yields 8× compression over the original proposal of compact blocks. An 80 times of data size reduction is thus realized.

## 2 Background

We now provide necessary background of this work including the details of the blockchain systems, and the throughput of blockchains.

### 2.1 Blockchain System

A blockchain is a distributed ledger that records all updates of the system state. Each operation of the update is known as a *transaction*. Multiple transactions are packed into a block, and blocks are linked with each other to construct a complete blockchain.

A blockchain system is maintained by many full nodes. Each full node holds a consistent replica of the blockchain. miner is a special full node, which creates new blocks to extend the blockchain. Different nodes are connected with each other and form the blockchain network.

Bitcoin is a typical blockchain. A transaction in Bitcoin transfers assets from one user to another. A transaction is broadcast into the blockchain network, and propagated among Bitcoin nodes using the gossip protocol. When a miner receives a new transaction, the miner will put it into a memory pool, which is a list used to temporarily store newly-received transactions. When creating a new block, the miner will select some transactions from the memory pool, and packed them into the block. The size of a block is 1–1.2 MB, and usually carries 1000–3000 transactions [2].

Different miners create a new block concurrently. Bitcoin introduces a mathematical puzzle (i.e., proof of work, PoW) to decide which block is appended to the blockchain. Miners compete with each other, and the first miner who solves the puzzle will broadcast its block into the network. When a full node receives this block, it will verify the solution and then append this block to the blockchain.

Due to the propagation delay, other miners may also work out the solutions before this new block are broadcast into the entire network. This will cause inconsistent ledgers among different full nodes because each full node may receive new blocks in different orders. This temporary inconsistency is known as a *fork*. Bitcoin uses Longest-Chain Rule to determine the canonical chain. Since a fork can compromise the security of Bitcoin, the difficulty of PoW is dynamically adjusted to control the frequency of new block generation. Currently a new block is generated every 10 minutes.

Our work is based on Bitcoin, but can also be applied to Ethereum and other blockchains.

## 2.2 Blockchain Throughput

The blockchain throughput, also known as TPS (Transactions Per Seconds), is the number of transactions recorded into the blockchain per unit time. Previous work [9] has revealed an estimating formula of the throughput.

$$TPS = \beta(\lambda) \cdot b \quad (1)$$

In Equation 1,  $b$  is the number of transactions carried by a block, and  $\beta$  is the rate of block addition to the canonical chain, which is positively related to the frequency of new block generation  $\lambda$ . Therefore, the throughput can be improved by:

1. Increasing  $b$ . We can compress the transaction size in each block, so a block can carry more transactions without causing pressure on the network bandwidth.
2. Improving  $\lambda$ . This can be achieved by reducing the difficulty of PoW. In order to reduce the probability of forks, the propagation delay of new blocks must be shortened. Research has shown that small-sized blocks can save the network bandwidth and thus shorten such delay [4, 5]. We can also compress the transaction size to construct smaller blocks.

In conclusion, the compression of transaction size can save the network bandwidth, and is an effective method of improving the blockchain throughput.

## 3 Related Work

Xthin [8] is the first scheme of transaction compression. A block in Xthin carries 256-bit TXIDs instead of complete transactions. The size of each block is thus reduced by around 10 times. BIP152 [3] uses compact blocks to compress the block size in Bitcoin, which uses the same method as Xthin. Such schemes dramatically accelerate the propagation of new blocks in the network. However, subsequent research has shown that there is still a large compression space for the block size.

On the basis of Xthin, Graphene [7] further compresses a Bitcoin block into 2.6 KB. However, it utilizes additional data structures including bloom filters or IBLTs (invertible bloom lookup tables). This adds the complexity to the implementations. Xthinner [10] uses the prefixes of TXIDs to represent transactions in each block, but it relies on the stack-based state machine to distinguish prefixes with unequal lengths.

## 4 Txilm Protocol

In this section, we first introduce the design of Txilm Protocol, and then model the probability of hash collisions. We further reduce this probability using the sorted transactions based on TXIDs. And finally, we simulate hash collisions to evaluate our probability models.

### 4.1 Protocol Design

Txilm derives from the compression of TXIDs. The 32-byte TXID is the SHA256 value of a transaction, which acts as a unique identifier of this transaction in the Bitcoin blockchain. Based on the TXID, the representation of a transaction can be further compressed by a short hash function:

$$\text{TXID-HASH} = \text{hash}(\text{TXID})$$

In the above equation,  $\text{hash}$  is a function that generates 32-bit to 64-bit hash values, such as CRC32, CRC40 or CRC64. Each new compact block includes only a list of TX-HASHes, ordered by the original list of transactions.

Ambiguity may occur with such a  $k$ -bit small-sized hash, which needs to be resolved by each full node. Once receiving a new block that includes the TX-HASH list from the sender, the receiver searches each received TX-HASH in the TXID list produced by its memory pool. For each TX-HASH, three cases may happen as follows:

1. Not found: There is no transaction in the memory pool that matches the received TX-HASH. The receiver will ask the sender or other peers for the missing TXID.
2. A single match is found: the TXID for the received TX-HASH is resolved.
3. Multiple matches are found: the receiver will collect all matched TXIDs as candidates for a 2nd-stage resolving.

Each block header carries the SHA256-Merkle root of all contained TXIDs. In the 2nd-stage, all combinations of candidates are iterated to recompute this Merkle tree. A correct combination will result in a matched TXID Merkle root with the one carried by the block header.

An optimization for the 2nd-stage is to add a lightweight pre-check before actually recomputing the Merkle root. We propose a lightweight Merkle tree for pre-check by replacing SHA256 with CRC32, which leads to a 4-byte root. When

creating a new block, the 4-byte CRC32-Merkle root is computed using the TXIDs and encoded into the block. This yields a  $40\times$  acceleration in searching the right combination.

If any of the combinations can not match the Merkle root in the block header, the receiver will fall back to ask the sender to transfer the complete TXID list of the block, which is described in the original BIP152 proposal. This situation happens when at least one TXID in the memory pool has the same TX-HASH in the received TX-HASH list, but the TXID is not the one that the block intends to include.

## 4.2 Probability of a Single Collision

Iterating the combination of candidates consumes a considerable amount of CPU time, and incurs additional latency. The feasibility of Txilm thus highly relies on the probability of hash collisions. We use the **single collision** to model this probability.

**Definition 1 (Single Collision)** We assume  $M$  is the set of all unconfirmed transactions contained in a memory pool, and  $b$  is an unconfirmed block. Given TX-HASH  $h \in b$  and its original transaction  $tx_1$ ,  $\exists tx_2 \in M \setminus b, tx_1 \neq tx_2$ , but  $\text{hash}(tx_2) = h$ .

In definition 1, a single collision can occur both within  $b$  or between  $b$  and  $M$ .  $P_{sc}$  can be formulated as the Generalized Birthday Problem [1]. We assume a memory pool  $M$  contains  $m$  transactions, and a new block  $b$  carries  $n$  TX-HASHes.  $M$  may in advance contain any of the transactions that  $b$  intends to carry, which will influence the value of  $P_{sc}$ . But in extreme case  $M$  contains none of the transactions, which will lead to the upper bound of  $P_{sc}$ . In this case the probability of a single collision is approximated as:

$$P_{sc} = 1 - \left[ 1 - \left( \frac{1}{2} \right)^k \right]^{\binom{mn + \frac{n^2}{2}}{2}} \quad (2)$$

## 4.3 Protocol Optimization

Txilm can be further optimized to reduce the probability of hash collisions. In this optimization, both the TX-HASHes in a new block and the transactions in the memory pools are sorted by the lexicographic order of TXIDs. Given a TX-HASH  $h$  in the block  $b$  with  $\text{TXID} = id$ , the candidate space of  $h$  will be narrowed to a range bound by  $id$  and its previous adjacent TXID in  $b$ , instead of the entire memory pool. This dramatically reduces the collision probability and the cost of resolving ambiguity, which allows even shorter hash with higher compression ratio.

We also use  $P_{sc}$  to evaluate the effect of this optimization. If each memory pool contains  $m$  transactions, and  $n$  TXIDs in a new block are evenly distributed in the sorted memory pool,

the size of the collision space will be reduced from  $m$  to  $\frac{m}{n}$ . The model for  $P_{sc}$  is therefore updated as:

$$P_{sc} = 1 - \left[ 1 - \left( \frac{1}{2} \right)^k \right]^m \quad (3)$$

## 4.4 Simulation

We simulated the hash collisions to evaluate above probability models. Sets  $M$  and  $b$  were defined to simulate the memory pool and new block respectively. We assume  $|M| = m$  and  $|b| = n$ . Our simulation set  $m = 1000$ , and  $k$  was iterated from 20 to 40. During each iteration of  $k$ , we simulated the computation and packing process of TX-HASHes for  $N = 100,000$  times, and counted  $T_k$ , the time of at least one single collision occurring between  $M$  and  $b$  or within  $b$ .  $P_{sc}$  is approximated as  $\frac{T_k}{N}$ .

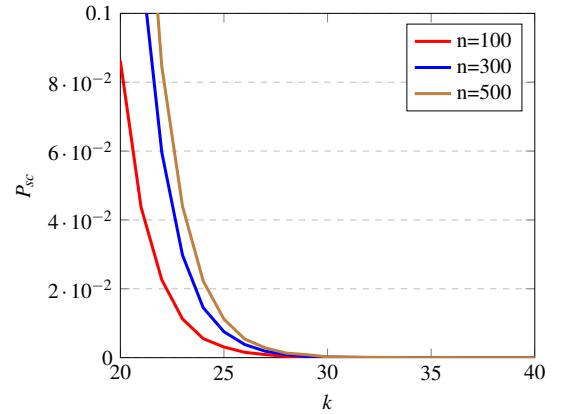


Figure 1:  $P_{sc}$  before Optimization

We first simulated the model of  $P_{sc}$  under the condition that above optimization was not applied. Figure 1 reveals the relation between  $P_{sc}$  and  $k$  with  $n = 100, 300, 500$  respectively. For each value of  $n$ ,  $P_{sc}$  decreases with the increase of  $k$ . Furthermore, it demonstrate that  $P_{sc}$  will increase with the block size  $n$ , since a large block are more likely to contain TX-HASHes that collide with other ones.

Figure 2 reveals the simulation results after the optimization. Our optimization achieves much smaller  $P_{sc}$  compared with the results in Figure 1. We recommend  $k = 32$  as a reasonable value with a better compression ratio and low collision rate. When  $m = 3,000$  and  $n = 200$ , a 32-bit TX-HASH makes  $P_{sc} = 0.00000072$ . This yields an 80 times of data size reduction compared with the original version of the blockchain.

## 5 Collision Attack

An attacker may create a new transaction with its TX-HASH matched to an existing transaction. A flood of such mali-

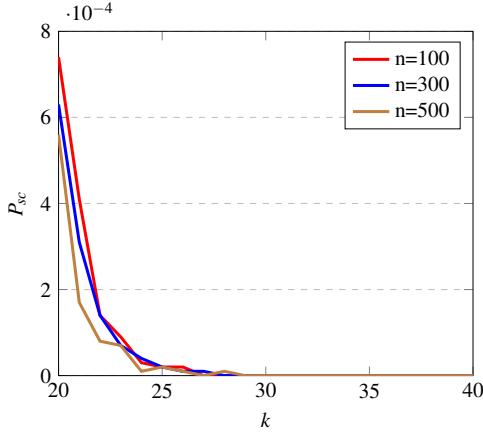


Figure 2:  $P_{sc}$  after Optimization

cious transactions for collisions can invalidate our collision probability analysis and make the verification of new blocks expensive, which eventually results in higher fork rate. We propose simple strategies to address this potential attack.

A simple strategy for defense is to introduce a salt while calculating the TX-HASH:

$$\text{TXID-HASH} = \text{hash}(\text{Salt} + \text{TXID})$$

The salt is specific to the block carrying these TX-HASHes, and is included in the encoded data (e.g., the block header). For example, just take the CRC32-Merkle root as the salt, or introduce another 4-byte field with random bits.

By introducing salts, the attacker cannot create malicious transactions that match existing transactions, until a new block carrying these existing transactions is generated. Malicious transactions are also unlikely to reach the entire network faster than a new block unless the attacker controls a large botnet.

In extreme cases, the massive collision attack is still possible regardless of the high cost. We require miners fall back to TXID list when the entire network is under attack. Such attack can be noticed by all miners when verifying new blocks. A miner can simply count the number of ambiguous TX-HASH per-block. If the counts are significantly higher than the expected value and forks are observed, the next block should fall back to TXID list.

## 6 Conclusion

This paper proposes Txilm, a protocol that compresses transaction representation in the blockchain. Each block in Txilm carries the short hashes of TXIDs (i.e., TX-HASHes) instead of complete transactions. When receiving a block from the peers, a full node can search the transactions in its local memory pool based on the TX-HASHes and resolve the hash collisions using Merkle root. We analyze the probability of hash collisions by simulation, and use the salted hash to protect against

potential attacks. Combined with the sorted transactions based on TXIDs, Txilm realizes 80 times of data size reduction and dramatically saves the network bandwidth. This will lead to a higher blockchain throughput.

In our future work, we will implement Txilm Protocol, and evaluate the effect of this protocol on the blockchain throughput by experiments.

## References

- [1] Wiki: Birthday problem. [https://en.wikipedia.org/wiki/Birthday\\_problem#Cast\\_as\\_a\\_collision\\_problem](https://en.wikipedia.org/wiki/Birthday_problem#Cast_as_a_collision_problem), 2018.
- [2] Block explorer. <https://www.blockchain.com/explorer>, 2019.
- [3] Matt Corallo. Bip152: Compact block relay. <https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki>, 2016.
- [4] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In *Financial Cryptography and Data Security*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- [5] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sep. 2013.
- [6] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, 2016. USENIX Association.
- [7] A. Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. Graphene: A new protocol for block propagation using set reconciliation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 420–428, Cham, 2017. Springer International Publishing.
- [8] Peter R. Rizun. Towards massive on-chain scaling: Block propagation results with xthin. [https://medium.com/@peter\\_r/towards-massive-on-chain-scaling-block-propagation-results-with-xthin-a0f1e3c23919](https://medium.com/@peter_r/towards-massive-on-chain-scaling-block-propagation-results-with-xthin-a0f1e3c23919), 2016.
- [9] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

- [10] Jonathan Toomim. Benefits of ltor in block entropy encoding. [https://medium.com/@j\\_73307/benefits-of-ltor-in-block-entropy-encoding-0r-8d5b77cc2ab0](https://medium.com/@j_73307/benefits-of-ltor-in-block-entropy-encoding-0r-8d5b77cc2ab0), 2018.