Txilm: Lossy Block Compression with Salted Short Hashing

Donghui Ding
ICT/CAS & University of Chinese Academy of Sciences

Xin Jiang
ICT/CAS & University of Chinese Academy of Sciences

Jiaping Wang
Monoxide Dev Team & ICT/CAS

Hao Wang
Monoxide Dev Team & ICT/CAS

Xiaobing Zhang
Monoxide Dev Team & ICT/CAS

Yi Sun
ICT/CAS & University of Chinese Academy of Sciences

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 and thus saves the bandwidth of the blockchain network. In this protocol, a block carries short hashes of `TXIDs` instead of complete transactions. Combined with the transaction list sorted by `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 possible attacks on Txilm.

1 Introduction

Blockchains have been applied to wide areas such as finance and supply chains. 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.

Aimed at the throughput problem of Bitcoin, compact blocks are proposed by BIP152. 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. BIP152 is safe because most transactions have already been stored in the memory pool of each Bitcoin node.

On the basis of BIP152, we propose Txilm, which further reduces the size of the transaction representation in a compact block to around 40 bits. This achieves $6.4\times$ compression over the original proposal of compact blocks. 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 using the transaction list sorted by `TXIDs`. Such optimization reduces the transaction representation in a block to 32 bits, which yields $8\times$ compression over the original proposal of compact blocks. An 80 times of data size reduction is thus realized.

2 Txilm Protocol

In this section, we first give the design of Txilm Protocol, and then evaluate the probability of hash collisions. We further reduce the probability using the sorted transaction list. And finally, we analyze the possible attacks on Txilm, and propose our defense strategies.

2.1 Protocol Design

Txilm derives from the compression of `TXIDs`. The 32-byte `TXID` is the SHA256 value of a transaction, which acts as the 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:

$$\texttt{TXID-HASH} = \texttt{h}(\texttt{TXID})$$

In the above equation, `h` 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-HASH`, ordered as 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 for recomputing 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× 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.

Iterating the combination of candidates consumes a considerable amount of CPU time, and incurs additional latency. The feasibility of this proposal thus highly depends on the probability of hash collisions, which is related to the length of the hash value (the k-bit) and also the size of the memory pool and block.

2.2 Probability of a Single Collision

A single collision is defined as the Case3 occurring at least once. Such collision can be within the TX-HASH list received, or the ones between the received list and the memory pool.

Given a TX-HASH with k bits, the collision rate is $\left(\frac{1}{2}\right)^k$. So the probability of a single collision occurring in a new block can be formulated as Generalized Birthday Problem. Assuming we have total m transactions in the memory pool and the new block carries n transactions, the probability of a single collision is approximated as:

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

We simulated the hash collisions to evaluate above probability model. In this simulation, k was iterated from 20 to 40. Two sets S_m and S_n were defined to simulate the memory pool and block respectively. We set $|S_m| = m$ and $|S_n| = n$, and filled S_m and S_n with k-bit random values. During each iteration of k, we refilled the values in S_m and S_n for $N = 100,000$ times, and counted T_k, the times of following two cases:

1. At least one value in S_m and S_n is equal.
2. S_n contains equal values.

Figure 1 reveals the relation between P_{sc} and k. We set $m = 3000$ and $n = 200$, and k on the horizontal axis ranges from 20 to 40. The dotted line is the measured frequency of at least one collision in each iteration of k, which is represented by $\frac{T_k}{N}$. Meanwhile, the solid line is the expected value of P_{sc} deduced from Equation 1. It demonstrates that the measured value is consistent with expected value. We recommend $k = 40$ as a reasonable value with better compression ratio and pretty low collision rate, which makes $P_{sc} = 0.00000056$.

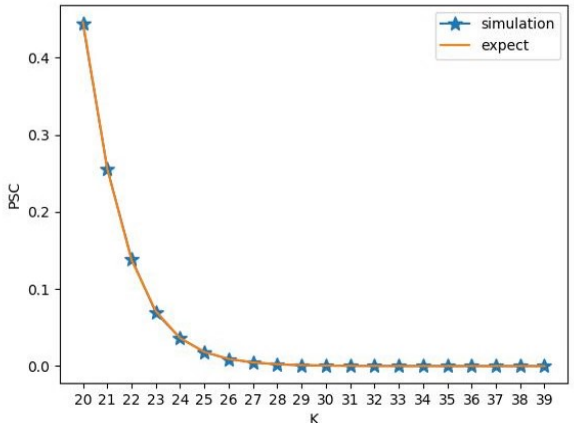

Figure 1: Relation between P_{sc} and k

2.3 Optimization by Sorted Transaction List

Txilm can be optimized by introducing the sorted transaction list. In this optimization, both the TX-HASHes in a new block and the transactions in the memory pool are sorted by the lexicographic order of TXIDs. The candidate space of any ambiguous TX-HASH will therefore be narrowed to a range bound by its previous and next TXID with resolved TX-HASH, instead of the entire memory pool. Assuming the newly confirmed TXIDs are evenly distributed in the sorted memory pool, the size of the potential collision space will be reduced from m to $\frac{m}{n}$. The collision within a block only occurs if the ambiguous TX-HASHes are adjacent after sorting. This dramatically reduces the collision probability and the cost of resolving ambiguity, which allows even shorter hash with higher compression ratio.

With this optimization, the probability model is updated as:

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

Given the same parameters including m, n and k, this model achieves a lower P_{sc} compared with Equation 1. When $m = 3000$ and $n = 200$, we recommend $k = 32$, which makes $P_{sc} = 0.00000072$. This yields a 80 times of data size reduction compared with the original version of the blockchain.

2.4 Collision Attack

An attacker may create a new transaction with its TX-HASH matched to an existing transaction. A flood of such malicious transactions for collisions can invalidate the collision probability analysis stated above and make the verification of new blocks expensive, which eventually results in higher fork rate. We propose simple strategies to address this attack model.

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

```
TXID-HASH = h( Salt + TXID )
```

The salt is specific to the block carrying those 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 even when existing transactions are known to all, 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.

3 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 tree. We analyze the probability of hash collisions by simulation, and use the salted hash to protect against possible attacks. Combined with the transaction list sorted by TXIDs, Txilm realizes 80 times of data size reduction and thus dramatically saves the network bandwidth.

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