```
BIP: 180
Layer: Peer Services
Title: Block size/weight fraud proof
Author: Luke Dashjr <luke+bip@dashjr.org>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0180
Status: Rejected
Type: Standards Track
Created: 2017-03-17
License: BSD-2-Clause
```

Abstract

A fraud proof that enables light clients to detect oversized (or overweight) blocks.

Copyright

Definitions

full tx size proof : SHA2 midstate and tail data proving the size of the full transaction data be

Specification

Proof format

- varint: ceil(log2(number of transactions in block))
- varint: number of size components in stripped-size proof
- foreach:
  - varint: ceil(log2(number of transactions represented by this size-component)) + 1
  - if zero:
    - (this indicates a full tx size proof)
    - 256-bit: SHA2 midstate up until just before the final SHA2 chunk
    - varint: total size of tx
    - uint8: size of final SHA2 chunk (0-55)
    - 0-55 bytes: final SHA2 chunk
  - if one or more:
    - (this indicates default tx size counting)
    - 256-bit: SHA2 hash of merkle link
- varint: number of size components in full-size proof (zero in case of a size-exceeded proof; non-zero for a weight-exceeded proof)
- foreach: (same as with stripped-size proof)

Proof verification

To verify an individual size proof:

1. Check that at least one size component is a full tx size proof. (At least one size component MUST be a full tx size proof.)
2. Determine the lower-bound number of transactions in the block (lowTx-Count). It is either `pow(ceil(log2(txcount)) - 1, 2)`, or the position of the last full tx proof (plus one, if using 0-based positions). Note that the last full tx proof from *either* of the size proofs (stripped-size and full-size) should be used here.
3. Calculate the lower-bound transaction-data size as the default size * lowTx-Count.
4. For each full tx size proof:
   (a) Subtract the default size it was presumed to consume, and add the claimed total size of tx.
   (b) Take the SHA2 midstate, and update it with the final SHA2 chunk (which needs to be padded, including with the total tx size). The final SHA2 hash is the transaction id (stripped-size proof) or hash (full-size proof).
5. For the full-size proof, replace the 60 byte default with any larger sizes proven from the stripped-size proof.
6. Build the merkle root, and compare it to the block header (stripped-size proof) or witness commitment (full-size proof). Ensure when building the merkle root, that there are no duplicate merkle links, and each merkle link claims to represent the correct number of represented transactions.
7. Add 80 bytes, plus the size of the tx-count varint, to the calculated lower-bound size.
8. The calculated size is returned as the lower-bound possible size of the block.

For the stripped-size proof, the default size of transactions is 60 bytes. For the full-size proof, it is the size established by the stripped-size proof.

To verify the complete weight proof:

1. Verify the stripped-size proof. Save the resulting lower-bound size (call it lowStrippedSize).
2. Verify the full-size proof. Save the resulting lower-bound size (call it lowFullSize).
3. Calculate minFullSize + (minStrippedSize * 3). This is the lower-bound block weight.
4. Compare the lower-bound block weight to the applicable block weight limit.

Network protocol

If a light client detects that one or more of its peers do not consider the block it knows to have the most work as their best block, it should inquire with all those peers for a fraud proof by sending a new message `getfraud`, with a block locator (between the last common block, and the presumed best tip) as the sole parameter (extra parameters should be ignored).

Compatible nodes will respond with a (new) `fraud` message, which has 2-3 parameters:

- uint256: The hash of the most recent block in the locator (or a parent thereof) that it has checked. In the event of an invalid block, this should be the exact invalid block's hash (post-invalid blocks should be treated as unchecked, even if the node has independently checked them for some reason).
- varint: Fraud proof type code
  - 0 = Block is valid
  - 1 = No fraud proof available
  - 2 = Size/weight exceeded
- (For type 2) the fraud proof

If none of the blocks in the locator are recognised, compatible nodes should send a `fraud` message with no parameters. (To avoid this outcome, clients may include a known-common block in the locator.)

In the event that the peer claims a block earlier than the client's tip is valid, the light client should prepare a new locator between that block and its tip, and rerequest `getfraud` until it has determined which block the peer rejects and why.

Once a block is proven to be invalid, the light client should never consider any blockchain including it as a candidate for the best chain. It should not recheck blocks known to be invalid, nor continue proving it from other nodes. (To avoid doubt: the user MAY be given the opportunity to override any rejections, but should be warned of the implications of doing so.)

If an invalid fraud proof is provided, the client SHOULD CONSIDER disconnecting and possibly banning the node providing it. However, if any change has been made to the size/weight limits, that should be taken into consideration (eg, if the limit increases, an innocent node may prove a size smaller than the limit).

Information

Creation of proofs

Proofs should ideally use the smallest amount of data required to prove excess of the limit. The most obvious mechanism in doing so, would be to include full tx size proofs for the largest transactions until the limit is exceeded. However, in some cases, a smaller size may be accomplished by collapsing more merkle links.

Because optimisation of proof size may be complicated, nodes are not required to implement it in any particular manner, so long as the proofs meet the requirements given above in Proof verification.

Motivation

Recently, there have been proposals for hardforks to increase the block size limit. While no consensus has been reached, proponents of these ideas often threaten and attempt to have miners force them through anyway. As things presently are, light clients cannot detect invalid blocks at all, and could be fooled into accepting an invalid chain created in such a manner. By supporting block size fraud proofs, light clients can protect their users from this form of unconsensual "hardfork" attempt.

Rationale

Why must a full tx size proof be included?

- This is necessary to establish that the claimed block transaction count is not inflated. Otherwise, a prover could claim any number of represented transactions for merkle links, and rely on the default size alone to exceed the limit.

How does the full tx size proof actually prove the size?

- The first step of SHA2 hashing is to transform the input data into chunks (per RFC 4634). The final chunk is required to include the absolute length of the input data at the end of the final chunk. Therefore, by committing to the midstate prior to the final chunk, and replaying only the final chunk, we can confirm that the claimed size matches the full transaction data being hashed.

How does this prove the block weight?

- The block weight defined by BIP 141 is the size of the block stripped of its segwit signatures times 3, plus the full size of the block. By proving lower-bound sizes of both the stripped block and the full block, a lower-bound weight can also be calculated.

Why is the number of transactions in the block represented as a log2?

- To avoid attacks that rely on fooling clients by claiming an amount they cannot verify.

Why does it matter if a full tx size proof is on the right side of a duplicate merkle link?

- We assume full tx size proofs show the number of transactions in the block. This assumption doesn't hold if the proof is provided on the right-hand side of duplicate links.

Why a fraud proof only for oversized/overweight blocks?

- While it is currently believed to be impossible to prove all invalid (or rather, won't-be-part-of-the-main-chain) blocks, there are regularly active proposals of miners attacking with simply oversized blocks in an attempt to force a hardfork. This specific attack can be proven, and reliably so, since the proof cannot be broken without also breaking the attempted hardfork at the same time.

Backwards compatibility

These fraud proofs protect only clients which use them. In non-attack scenarios, they are unnecessary and clients supporting them will otherwise behave as any other.

Reference implementation

TODO