Bitcoin Block Propagation with IBLT

Rusty Russell

Code Contributor, Blockstream
rusty@blockstream.com/*rusty@rustcorp.com.au*

The Problem

- Blocks are transmitted in their entirety.
  - In parallel to all peers.
  - 1MB blocks, 8 peers, 1Mbit → 66.8 – 76.4 seconds
- Miners can solve this easily by all centralizing!

The Opportunity

- Under *normal* circumstances, most peers already know many of the transactions.

The Opportunity

- Under *normal* circumstances, most peers already know many of the transactions.
- Doesn't help for worst case!

The Opportunity

- Under *normal* circumstances, most peers already know many of the transactions.
- Doesn't help for worst case!
- And we want to avoid adding round trip latency...

First Attempt

- Gavin Andresen's 'O(1) Block Propagation' gist:
  - https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2

First Attempt

- Gavin Andresen's 'O(1) Block Propagation' gist:
  - https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2

- Miners use Invertable Bloom Lookup Table to encode block for transmission

http://whitehairedoldgrouch.tumblr.com/post/833680337/mmmblts CC-BY-2.0

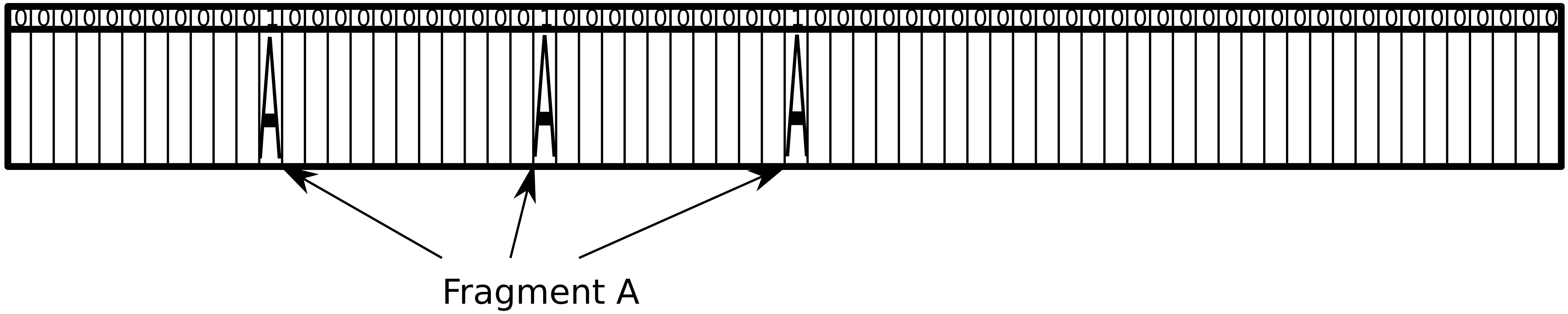
IBLT: Background

- Slice transaction into equal fragments:

```
struct fragment {
    u8 id[6];
    u16 index;
    u8 frag[8];
} key;
```

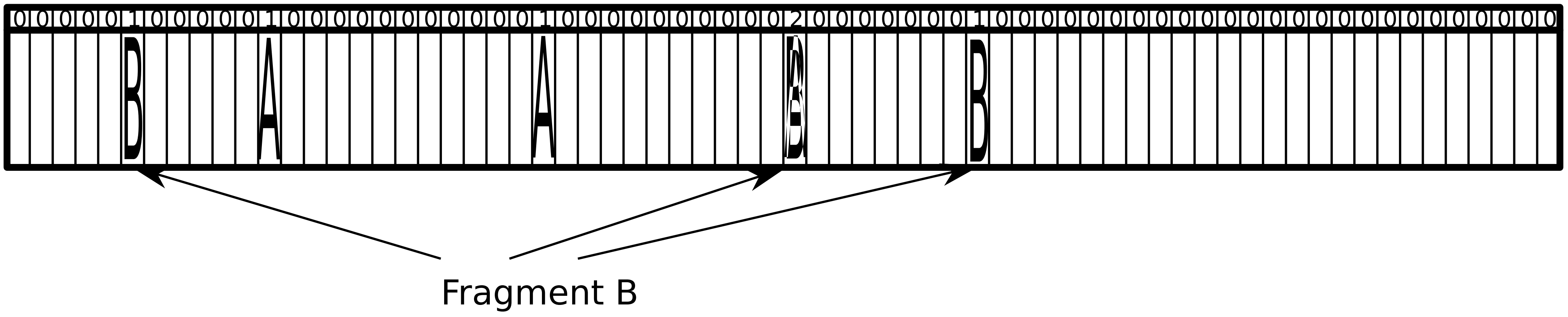
IBLT: Background

- Use three hash functions to place it into buckets:
  - Increment counter for the bucket.
  - XOR in the fragment



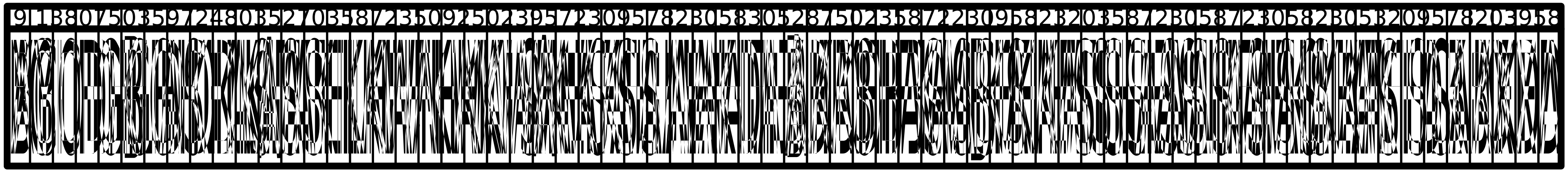
IBLT: Background

- Repeat for other fragments



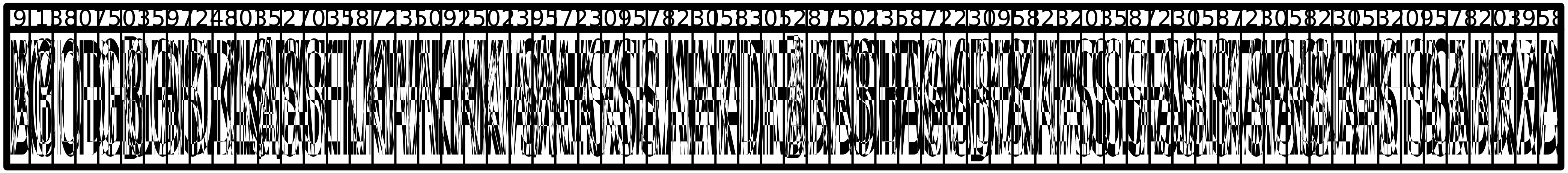
IBLT: Background

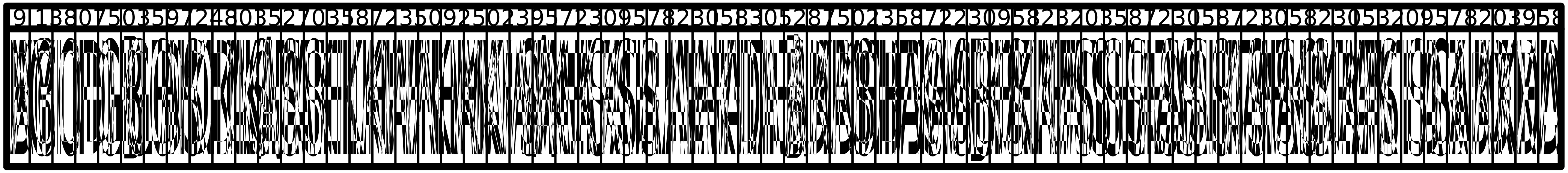
- Repeat for other fragments



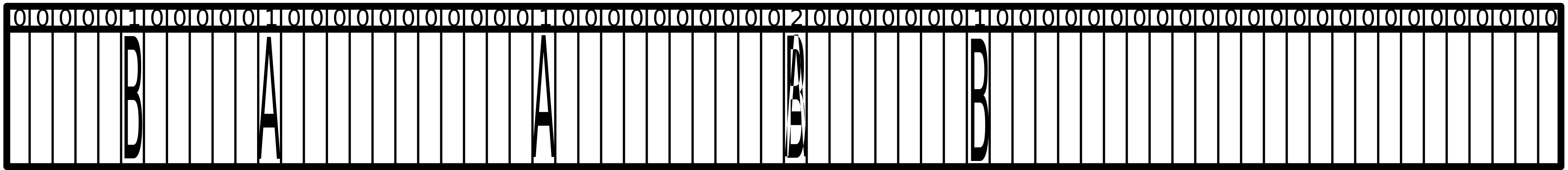
IBLT: Background

- Send to peer
- Peer creates equivalent IBLT
- Calculates difference





Subtract counters, XOR fragments:



IBLT: Background

- Buckets with -1: tx not in block.
  - → Eliminate all tx fragments from IBLT.
- Buckets with 1: unknown tx in block
  - → Remove, reassemble tx once all frags recovered
- If we end up with empty IBLT, try to form block.

Minor Improvements

```
struct fragment {
        u8 id[6];
        u16 index;
        u8 frag[8];
} key;
```

- Use siphash not SHA256 for id (v. fast)
- Offset index by hash of id (decode ordering)
- Larger than 8 byte fragments.
- Fewer bits (than 32) for bucket counter.

(Thanks to Kalle Rosenbaum for discussion)

Peer-to-peer IBLT

- Creating an IBLT is *fast*:
  - Create frag ids from secret + txid for all txs in mempool (+ any other known txs).
  - XOR txs into IBLT.
- Let's use this between peers!
  - Thanks Pieter Wuille

IBLT: Scaling

IBLT: Scaling

- Scales by *differences in mempool*
  - With some encoding penalty (1-2.2x)
- Implies that it scales with tx bitrate.

IBLT: Centralization?

- Pressure on miners to minimize mempool differences.
  - Implications for censorship.

IBLT: Centralization?

- Pressure on miners to minimize mempool differences.
  - Implications for censorship.
- Tradeoff:
  - We need to indicate which mempool txs are likely to be in block.
  - Need a compact heuristic to represent block txs
  - Try not to make “reasonable variations” cost too much.

IBLT: Centralization?

- Send “minimum satoshi per byte”.
  - Assumes miners are basically profit-maximizing.

IBLT: Centralization?

- Send “minimum satoshi per byte”.
  - Assumes miners are basically profit-maximizing.
- Add “txs which are below that but included”
- Add “txs which are above that but excluded”

IBLT: Centralization?

- Send “minimum satoshi per byte”.
  - Assumes miners are basically profit-maximizing.
- Add “txs which are below that but included”
- Add “txs which are above that but excluded”
  - These two can be compactly represented as bit prefixes
  - O(#txs-in-mempool) bits
  - eg. 20 bits for 1M txs in mempool.

Rough Results

https://github.com/rustyrussell/bitcoin-corpus

  - 1 week mempool data of 4 nodes on Digital Ocean

- Pretend they are peers

Rough Results

- 128 byte fragment size
  - → Best possible case is 15.4MB instead 482.3MB

Rough Results

- 128 byte fragment size
  - → Best possible case is 15.4MB instead 482.3MB
- Good: Block 352778 (999770 bytes):
  - – 999599 bytes known, 0 bytes unknown.
  - – 1273086 bytes in mempool.
  - – Best possible total tx bytes: 1898, 1898, 4244

Rough Results

- 128 byte fragment size
  - → Best possible case is 15.4MB instead 482.3MB
- Good: Block 352778 (999770 bytes):
  - 999599 bytes known, 0 bytes unknown.
  - 1273086 bytes in mempool.
  - Best possible total tx bytes: 1898, 1898, 4244
- Bad: Block 352737 (99749 bytes)
  - 15371 bytes known, 84202 bytes unknown.
  - 137660 bytes in mempool.
  - Best possible total tx bytes: 112319, 112319, 112319

Canonical Block Ordering: by Fee

- IBLT doesn't include tx order.
  - Gavin suggested an arbitrary tx order
- Order by fee-per-kbyte:
  - Plus commitment to minfee and # txs below & above provides some fee determination for SPV

Canonical Block Ordering: by Fee

- IBLT doesn't include tx order.
  - Gavin suggested an arbitrary tx order
- Order by fee-per-kbyte:
  - Plus commitment to minfee and # txs below & above provides some fee determination for SPV
- Also helps “weak”-block propagation idea
  - Nodes would send blocks which reach 1/20th target
  - Net encoding could refer to previous weak blocks.
  - Most efficient if can use ranges
    - Fee-per-byte most likely to be contiguous.

Conclusion: Testing

- Test without miner support:
  - Sending accompanying ordering information.
  - Guesstimate minfee.
  - Use feedback from previous blocks to estimate how “in-sync” mempools are for each peer.
  - Combine with total unknown txsize for this peer to estimate appropriate IBLT size.
  - Aim for 95% chance of reconstruction.