Graphene: A New Protocol for Block Propagation Using Set Reconciliation

A. Pinar Ozisik† Gavin Andresen George Bissias† Amir Houmansadr† Brian N. Levine†
†College of Information and Computer Sciences, UMass Amherst

1 INTRODUCTION

We propose an efficient method of announcing new blocks called *Graphene*. This document summarizes a more detailed description of Graphene that has been published previously [7] and includes additional experiments.

Graphene blocks are a fraction of the size of related methods, such as Compact Blocks [3] and Xtreme Thinblocks [9]. For example, in a detailed example below, we show that a 17.5 KB Xtreme Thinblock can be encoded in 10 KB with Compact Blocks, and encodedin 2.6 KB with Graphene. In simulations, we find that Graphene encodes information in about 10% of the space of Compact Blocks. We use a novel interactive combination of Bloom filters [2] and Invertible bloom lookup tables (IBLTs) [5], providing an efficient solution to the problem of set reconciliation in Bitcoin's p2p network.

Block announcements are validated using the transaction content comprising the block. However, it is likely that the majority of peers have already received these transactions, and they only need to discern them from those in their mempool. In principle, a block announcement needs to include only the IDs of those transactions. For example, Corallo's *Compact Block* design [3] significantly reduces block size by including a transaction ID list, though the cost is increased coordination between peers. *Xtreme Thinblocks* [9] works similarly to Compact Blocks but has greater data overhead. Specifically, if an `inv` is sent for a block that is not in the receiver's mempool, the receiver sends a Bloom filter of her IDpool along with the request for the missing block. As a result, Xtreme Thinblocks are larger than Compact Blocks. The community has discussed in forums the use of IBLTs (without Bloom Filters) for reducing block announcements [1, 8], but these schemes have not been formally evaluated and are less efficient than our approach. Our method is novel; we have proven and demonstrated that it is smaller than all of these recent works, and still requires 3 messages between sender and receiver for coordination: an `inv`, a `getdata`, and a response.

PROTOCOL 1: Graphene

1: Sender: Sends *inv* for a block.
2: Receiver: Requests unknown block; includes count of txns in her IDpool, m.
3: Sender: Sends Bloom filter $\mathcal{S}$ and IBLT $\mathcal{I}$ (each created from the set of n txn IDs in the block) and essential Bitcoin header fields. The FPR of the filter is $f = \frac{a}{m-n}$, where $a = n/(c\tau)$.
4: Receiver: Creates IBLT $\mathcal{I}'$ from the txn IDs that pass through $\mathcal{S}$. She decodes the *subtraction* [4] of the two blocks, $\mathcal{I} \triangle \mathcal{I}'$.

Figure 1: A summary of the Graphene protocol.

2 THE GRAPHENE PROTOCOL

Unlike other approaches, Graphene never sends an explicit list of transaction IDs. Instead it sends a small Bloom filter and a very small IBLT. The intuition behind Graphene is as follows; a summary appears in Figure 1.

The sender creates an IBLT $\mathcal{I}$ from the set of transaction (txn) IDs in the block. To help the receiver create the same IBLT (or similar), he also creates a Bloom filter $\mathcal{S}$ of the transaction IDs in the block. The receiver uses $\mathcal{S}$ to filter out transaction IDs from her pool of received transaction IDs (which we call the IDpool) and creates her own IBLT $\mathcal{I}'$. She then attempts to use $\mathcal{I}'$ to *decode* $\mathcal{I}$, which, if successful, will yield the transaction IDs comprising the block. The number of transactions that falsely appear to be in $\mathcal{S}$, and therefore are wrongly added to $\mathcal{I}'$, is determined by a parameter controlled by the sender. Using this parameter, he can create $\mathcal{I}$ such that it will decode with very high probability.

In sum, the Bloom filter from the sender allows the receiver to determine which transactions from its mempool are in the block. Other approaches require a much larger Bloom filter to keep the false positive rate small; in Graphene, the Bloom Filter FPR is high because the IBLT recovers any mistakes made. Similarly, if only the IBLT was used, it would be much larger than our use of the two mechanisms.

A Bloom filter is an array of x bits representing y items. Initially, the x bits are cleared. Whenever an item is added to the filter, k bits, selected using k hash functions, in the bit-array are set. The number of bits required by the filter

is $x = y\frac{-\ln(f)}{\ln^2(2)}$, where f is the intended false positive rate (FPR). For Graphene, we set $f = \frac{a}{m-n}$, where a is the expected difference between $\mathcal{I}$ and $\mathcal{I}'$. Since the Bloom filter contains n entries, and we need to convert to bytes, its size is $\frac{-\ln(\frac{a}{m-n})}{\ln^2(2)}\frac{1}{8}$. It is also the case that a is the primary parameter of the IBLT size. IBLT $\mathcal{I}$ can be decoded by IBLT $\mathcal{I}'$ with very high probability if the number of cells in $\mathcal{I}$ is d-times the expected symmetric difference between the list of entries in $\mathcal{I}$ and the list of entries in $\mathcal{I}'$. In our case, the expected difference is a, and we set $d = 1.5$ (see Eppstein et al. [4], which explores settings of d).

Each cell in an IBLT has a *count*, a *hash* value, and a stored *value*. (It can also have a key, but we have no need for a key). For us, the count field is 2 bytes, the hash value is 4 bytes, and the value is the last 5 bytes of the transaction ID (which is sufficient to prevent collisions). In sum, the size of the IBLT with a symmetric difference of a entries is $1.5(2+4+5)a = 16.5a$ bytes. Thus the total cost in bytes, T, for the Bloom filter and IBLT are given by $T(a) = n\frac{-\ln(f)}{c} + a\tau = n\frac{-\ln(\frac{a}{m-\mu})}{c} + a\tau$, where all Bloom filter constants are grouped together as $c = 8\ln^2(2)$, and we let the overhead on IBLT entries be the constant $\tau = 16.5$.

To set the Bloom filter as small as possible, we must ensure that the FPR of the filter is as high as permitted. If we assume that all `inv` messages are sent ahead of a block, we know that the receiver already has all of the transactions in the block in her IDpool (they need not be in her mempool). Thus, $\mu = n$; i.e., we allow for a of $m-n$ transactions to become false positives, since all transactions in the block are already guaranteed to pass through the filter. It follows that

$$T(a) = n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau. \quad (1)$$

Taking the derivative with respect to a, Eq. 1 is minimized when when $a = n/(c\tau)$.

Actual implementations of Bloom filters and IBLTs involve several (non-continuous) ceiling functions such that we can re-write:

$$T(a) = \left(\lceil\ln(\frac{m-n}{a})\rceil\left\lceil\frac{n\ln(\frac{m-n}{a})}{\lceil\ln(\frac{m-n}{a})\rceil\ln^2(2)}\right\rceil\right)\frac{1}{8} + \lceil a\rceil\tau. \quad (2)$$

The optimal value of Eq. 2 can be found with a simple brute force loop. In practice, a for-loop brute-force search for the lowest value of a is almost no cost to perform, and we do so in our simulations.

Due to the randomized nature of an IBLT, there is a very small but non-zero chance that it will fail to decode. In that case, the sender resends the IBLT with double the number of cells (which is still very small). In our simulations, which encoded real IBLTs and Bloom filters, this doubling was sufficient for the incredibly few IBLTs that failed.

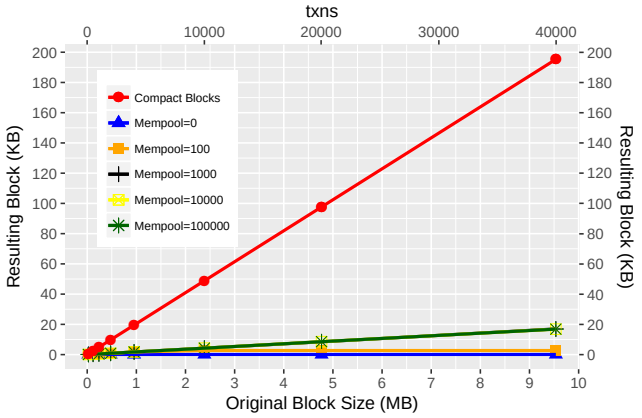

Figure 2: A comparison of Graphene and Compact blocks. Mempools are expressed in number of transactions (not bytes) above and beyond the block itself.

2.1 Comparison to Other Approaches

We have shown previously that Graphene is strictly more efficient than Compact Blocks through analysis [7]. For illustrative purposes, here is an example.

A receiver with an IDpool of $m = 4000$ transactions makes a request for a new block that has $n = 2000$ transactions. The value of a that minimizes the cost is $a = n/(c\tau) = 31.5$. The sender creates a Bloom filter $\mathcal{S}$ with false positive rate $f = \frac{a}{m-n} = 31.5/2000 = 0.01577$, with total size of $2000 \times \frac{-ln(0.01577)}{c} = 2.1$ KB. The sender also creates an IBLT with a cells, totaling $16.5a = 521B$. In sum, a total of $2160B + 521B = 2.6$ KB bytes are sent over the network. The receiver creates an IBLT of the same size, and using the technique introduced in Eppstein et al. [4], the receiver subtracts one IBLT from the other before decoding. In comparison, for a block of n transactions, Compact Blocks costs $2000 \times 5B = 10$ KB, over 3 times the cost of Graphene.

We also note that Xtreme Thinblocks [9] are strictly larger than Compact Blocks since they contain all IDs and a Bloom filter, and therefore Graphene performs strictly better than Xtreme Thinblocks as well.

Simulations. Figure 2 shows the results of a simulation of Compact Blocks and Graphene. The simulation is comprised of many trials. Each trial takes as input a block size (in terms of transactions) and the mempool size above and beyond the transactions in the block. Each protocol is executed and the number of bytes required is recorded. Because Bloom filters and IBLTs are probabilistic mechanisms, the simulation uses the real data structures to ensure the results are accurate. Each point on the plot is the mean of hundreds of simulations at that point. Error bars are too small to be shown.

As the figure shows, Graphene is consistently 1/10 of the cost of Compact block or less, depending on the mempool size. Graphene's size is dependent on the mempool size, but the growth is extremely slow.

2.2 Ordered blocks

Graphene does not specify an order for transactions in the blocks, and instead assumes that transactions are sorted by ID. Bitcoin requires transactions depending on another transaction in the same block to appear later, but a canonical ordering is easy to specify. If a miner would like to order transactions with some proprietary method (e.g., [6]), that ordering would be sent alongside the IBLT. For a block of n items, in the worst case, the list will be $n \log_2(n)$ bits long. Even with this extra data, our approach is much more efficient than Compact Blocks. In terms of the example above, if Graphene was to impose an ordering, the additional cost for $n = 2000$ transactions would be $n \log_2(n)$ bits = $2000 \times log_2(2000)$ bits = 2.74 KB. This increases the cost of Graphene to 5.34 KB, still almost half of Compact Blocks.

REFERENCES

[1] Gavin Andresen. O(1) Block Propagation. https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2, August 2014.

[2] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[3] Matt Corallo. Bip152: Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, April 2016.

[4] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the Difference?: Efficient Set Reconciliation Without Prior Context. In *ACM SIGCOMM*, 2011.

[5] M.T. Goodrich and M. Mitzenmacher. Invertible bloom lookup tables. In *Conf. on Comm., Control, and Computing*, pages 792–799, Sept 2011.

[6] Timo Hanke. A Speedup for Bitcoin Mining. http://arxiv.org/pdf/1604.00575.pdf (Rev. 5), March 31 2016.

[7] A. Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Neil Levine. Graphene: A New Protocol for Block Propagation Using Set Reconciliation. In *Proc. of International Workshop on Cryptocurrencies and Blockchain Technology (ESORICS Workshop)*, September 2017.

[8] Rusty Russel. Playing with invertible bloom lookup tables and bitcoin transactions. http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html, Nov 2014.

[9] Peter Tschipper. BUIP010 Xtreme Thinblocks. https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/, Jan 2016.