Graphene: A New Protocol for Block Propagation Using Set Reconciliation

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

Abstract. *We devise a novel method of interactive set reconciliation for efficient block distribution. Our approach, called Graphene, couples a Bloom filter with an IBLT. We evaluate performance analytically and show that Graphene blocks are always smaller. For example, while a 17.5 KB Xtreme Thinblock can be encoded in 10 KB with Compact Blocks, the same information can be encoded in 2.6 KB with Graphene. We show in simulation that Graphene reduces traffic overhead by reducing block overhead.*

1 Introduction

Blockchain-based currencies [8], such as Bitcoin and Ethereum, have seen widespread adoption despite several limitations not present in traditional financial systems, such as credit cards or cash. Network delays and overhead are noticeable in blockchains. Once discovered, the propagation delay and network cost for distributing new blocks is dependent on their size, and reducing either delay or traffic is desirable.

We contribute an extremely efficient method of announcing new blocks called *Graphene*. Our protocol is applicable to a variety of blockchain-based network protocols, such as Bitcoin, Ethereum [5], Litecoin (https://litecoin.org), and Zerocash [10]. Our blocks are a fraction of the size of related methods, such as Compact Blocks [3] and Xtreme Thinblocks [11]. For example, while a 17.5 KB Xtreme Thinblock can be encoded in 10 KB with Compact Blocks, the same information can be encoded in 2.6 KB with Graphene. We use a novel interactive combination of Bloom filters [2] and IBLTs [6], providing an efficient solution to the problem of set reconciliation in the p2p network. We evaluate performance analytically and empirically via a detailed network simulation. Graphene reduces traffic overhead to about 60% compared to using Compact Blocks if blocks are sent every 2.5 minutes; Ethereum would see higher gains from Graphene, and the gains for Bitcoin would be lower, since blocks are sent more and less frequently, respectively.

2 Background

In this section, we review the operation of IBLTs and summarize related work.

Non-first authors are listed alphabetically. Supported in part by an equipment grant from the Collaborative R&D Fund managed by the Massachusetts Technology Collaborative.

Overview of IBLTs. We make use of Invertible Bloom Lookup Tables (IBLTs) [6], which is an efficient data structure for *set reconciliation* between two peers. Like Bloom filters [2], IBLTs allow two parties to determine, with high probability, which values from a set they share in common. But unlike Bloom filters, IBLTs enable the recovery of any missing values, which are assumed to be of fixed size and encoded as binary strings. Key-value pairs can be inserted, retrieved and deleted like an ordinary hash table. An IBLT consists of m entries, each storing a `count`, a `keySum`, and a `valueSum`, all initialized to zero.

A new value v is inserted into location $i = h(v)$ based on the hash of its value such that $i < m$. At entry i, all three fields are incremented or xored. IBLTs use $k > 1$ hash functions to store each value in k entries, which we collectively call a value's *entry set*. If table space is sufficient, then with high probability for at least one of the k entries, $\texttt{count} \equiv 1$.

Suppose that two peers each have a list of values, V and V', respectively, such that the difference is expected to be small. The first peer constructs an IBLT L (with m entries) from V. The second peer constructs V' from L' (also having m entries). Eppstein et al. [4] showed that a cell-by-cell difference operator can be used to efficiently compute the symmetric difference $L \triangle L'$. For each pair of fields (f, f'), at each entry in L and L', we compute either $f \oplus f'$ or $f - f'$ depending on the field type. When $|\texttt{count}| \equiv 1$ at any entry, the corresponding value can be recovered. Peers proceed by removing the recoverable key-value pair from all entries in the value's entry set. This process will generally produce new recoverable entries, and continues until nothing is recoverable.

Related Work. The main limitation we are addressing with Graphene is the inefficiency of blockchain systems in disseminating block data. A block announcement must be validated using the transaction content comprising the block. However, it is likely that the majority of the 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, and accordingly, Corallo's *Compact Block* design [3] — which has been recently deployed — significantly reduces block size by including a transaction ID list at the cost of increasing coordination to 3 roundtrip times. We further detail Compact Block's operation in Section 3 and compare it quantitatively in Section 4. *Xtreme Thinblocks* [11], an alternative protocol, 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 but require just 2 roundtrip times. Relatedly, the community has discussed in forums the use of IBLTs (alone) for reducing block announcements [1,9], but these schemes have not been formally evaluated and are less efficient than our approach. Our novel method, which we prove and demonstrate is smaller than all of these recent works, requires just 2 roundtrip times for coordination.

3 Graphene: Efficient Block Announcements

In this section, we detail *Graphene*, where a receiver learns the set of specific transaction IDs that are contained in a (pending or confirmed) block containing n transactions. Unlike other approaches, Graphene never sends an explicit list of transaction IDs, instead it sends a small Bloom filter and a very small IBLT.

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, $\mathcal{I} \triangle \mathcal{I}'$.
```

The protocol. The intuition behind Graphene is as follows. 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.

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

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

Taking the derivative w.r.t. a, Eq. 1 is minimized[1] when when $a = n/(c\tau)$.

Due to the randomized nature of an IBLT, there is a 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, presented in the next section, this doubling was sufficient for the incredibly few IBLTs that failed.

PROTOCOL 2: CompactBlocks

```
1: Sender:    Sends inv for a block that has $n$ txns.
2: Receiver:  If block is not in mempool, requests compact block.
3: Sender:    Sends the block header information, all txn IDs in the block and any full
   txns he predicts the sender hasn't received yet.
4: Receiver:  Recreates the block and requests missing txns if there exist any.
```

Comparison to Compact Blocks. Compact Blocks [3] is to our knowledge the best-performing related work. It has several modes of operation. We examined the *Low Bandwidth Relaying* mode due to its bandwidth efficiency, which operates as follows. After fully validating a new block, the sender sends an `inv`, for which the receiver sends a *getdata* message if she doesn't have the block. The sender then sends a *compact block* that contains block header information, all transaction IDs (shortened to 5 bytes) in the block, and any transactions that he predicts the receiver does not have (e.g., the coinbase). If the receiver still has missing transactions, she requests them via an `inv` message. Protocol 2 outlines this mode of Compact Blocks. The main difference between Graphene and Compact Blocks is that instead of sending a Bloom filter and an IBLT, the sender sends block header information and all shortened transaction IDs to the receiver.

A detailed example of how to calculate the size of each scheme is below; but we can state more generally the following result. For a block of n transactions, Compact Blocks costs $5n$ bytes. For both protocols, the receiver needs the `inv` messages for the set of transactions in the block before the sender can send it. Therefore, we expect the size of the IDpool of the receiver, m, to be constrained

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

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

The optimal value of Eq. 2 can be found with a simple brute force loop. We compared the value of a picked by using $a = n/(c\tau)$ to the cost for that a from Eq. 2, for valid combinations of $50 \le n \le 2000$ and $50 \le m \le 10000$. We found that it is always within 37% of the cost of the optimal value from Eq. 2, with a median difference of 16%. 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.

such that $m \geq n$. Assuming that $m > 0$ and $n > 0$, the following inequality must hold for Graphene to outperform Compact Blocks:

$$n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau < 5n \tag{3}$$

$$n > {}^{m}/_{1287670} \tag{4}$$

In other words, Graphene is strictly more efficient than Compact Blocks *unless* the set of unconfirmed transactions held by peers is 1,287,670 times larger than the block size (e.g., over 22 billion unconfirmed transactions for the current block size.) Finally, we note that Xtreme Thinblocks [11] 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. In Section 4, we provide specific empirical results from network simulation, where we use real IBLTs and Bloom filters to evaluate Graphene and Compact Blocks.

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 $f = \frac{a}{m-n} = 31.5/2000 = 0.01577$, 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. 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.

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., [7]), 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.

4 Evaluation

Our evaluation addresses the following question: What is the reduction in traffic from using Graphene for block announcements compared to Compact Blocks?

Simulator assumptions. Our evaluations are based on a detailed, custom blockchain simulator using a Python-based discrete event simulator package. Our simulation models the propagation of messages across network links (ignoring effects from variable network bandwidth, TCP, etc.). Nodes accurately model any part of typical blockchain operation necessary for evaluating our metrics,

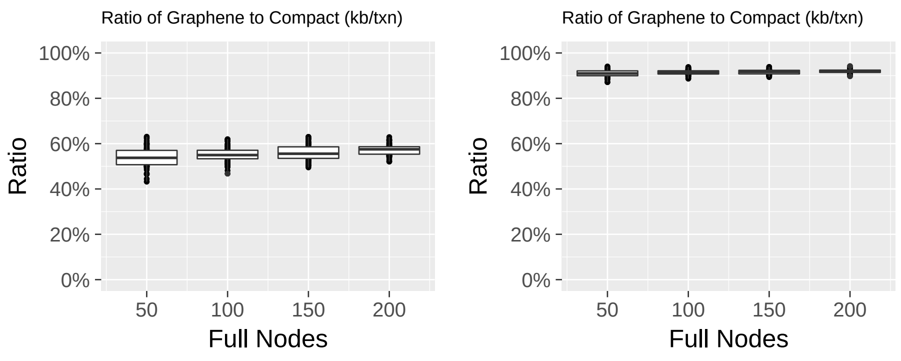

Fig. 1: When the current topology is used, Graphene reduces traffic to 60% of the cost of Compact Blocks (or to 10% for total traffic, which includes transaction data).

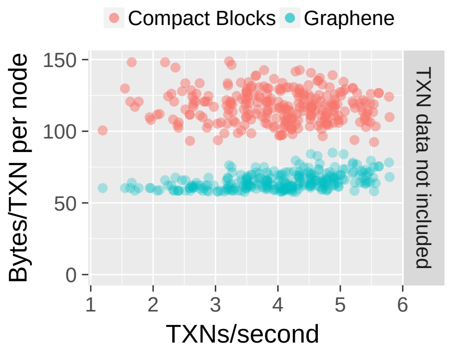

Fig. 2: Traffic sent by Graphene and Compact Blocks, where each trial's transaction rate is the independent variable. Transparency reveals some over-plotting in this scatterplot.

including maintaining a mempool, the blockchain and its forks, and using realistic signaling.

For Graphene and Compact Blocks, our simulator creates and decodes real Bloom filters and IBLTs, rather than merely estimating whether they might decode or return any false positives. If these data structures fail due to random chance, the nodes recover within the simulation. Because our simulation models detailed signaling and is written in a high-level language, our evaluations are based on a modest number of peers. Since our goal is a comparison between two choices, we expect that our results are representative of larger-scale scenarios.

A challenging parameter to set is the number of transactions per second offered to the network by peers. Our approach is to create kernel density estimates (KDEs) from the transaction generation patterns of real world peers. To that end, we gathered data for all Bitcoin transactions during a three-month period from http://blockchain.info. Each transaction in the dataset is labeled with an IP associated with the peer believed to have generated it, as well as the time it was released to the network. For each peer, we normalized the release times by the time of the day in which they were released. We then constructed the KDE for each peer using these normalized transactions times and gaussian kernels with one

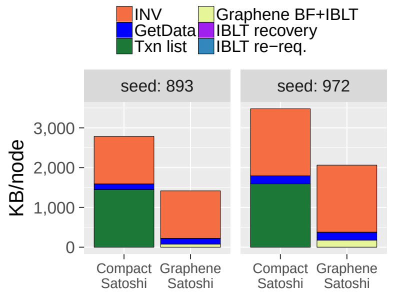

Fig. 3: A comparison of traffic, by message type, for two specific seeds for Graphene and Compact Blocks. N.b., traffic does not include transaction data.

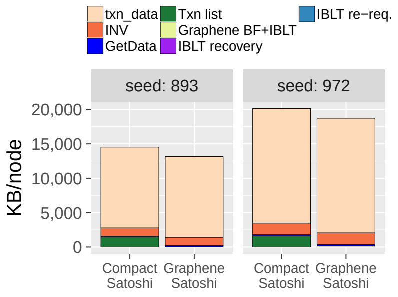

Fig. 4: Traffic by message type, for two specific seeds. Figure 3 shows the same plot without transaction data included.

hour bandwidth. The KDE for a given peer represents a probability distribution from which we can draw transactions over the course of a simulated day. For each peer in the simulator, we randomly select one of the KDEs corresponding to a real world peer. Because these distributions have been generated from real data, they are a good approximation of the activity of real peers over the average one-day interval. On the other hand, this approach is not able to model days of the week or seasonal phenomena in transaction creation times.

Results. Each simulation is configured to use the following parameters: *(i)* Topology: a high-degree p2p graph topology. *(ii)* Block Protocol: *Compact Blocks*; or our *Graphene* protocol. *(iii)* Block capacity: 2,000 transactions. *(iv)* Full nodes: 50, 100, 150, or 200 peers. In all, we ran 8 combinations of parameters, and we ran each combination with 67 different seeds; all told, we completed 536 simulations. The seeds determined the number of transactions per second (by sampling our KDE, as described above), and the interarrival of transactions and blocks. In all simulations, we used 6 miner nodes, representing 6 mining pools. Each simulation was equivalent to 120 minutes; in sum, we simulated about 45 days of blockchain operation. Blocks are generated every 2.5 minutes, like Litecoin; our results would show Graphene to have significantly greater savings if blocks were every 15 seconds (like Ethereum), and show significantly smaller savings if blocks were every 10 minutes (like Bitcoin).

Our main results are shown in Fig. 1, where we evaluated the total bandwidth ratio of Graphene to Compact Blocks, as a function of the number of nodes in the network. Since each run is a different number of KBs, we compare the ratio of an exact set of parameters (including the seed), varying only the protocol. Boxplots show the distribution of results across all trials. Fig. 1(left) shows that Graphene reduces traffic to 60% of the cost of using Compact Blocks. Note that gains reduce to 10% (i.e., are 90% of Compact Blocks) when transaction data is also included because they account for the largest portion of network traffic. However, as the number of full nodes increases along the x-axis, the ratio of total traffic in the network remains steady, suggesting that our results are representative of larger networks.

We also evaluated the sum number of bytes per message type for two example seeds, and details appear in Fig. 3. We saw that the amount of data used by Compact Blocks is much greater than Graphene's use of a Bloom filter and an IBLT. In Fig. 2, we also grouped our larger set of results according to transactions-per-second, and found that Compact Blocks generates a wide range of bytes-per-transaction, even at the lowest transactions-per-second rate. In contrast, Graphene is both more efficient and stable as load changes. Even when more transactions are generated, *Graphene uses less traffic* because the difference between the IDpool (of size m) and the block (of size n) is small, perhaps even zero, causing both its Bloom filter and IBLT to be negligible in size — see Eq 1.

5 Conclusion

We presented Graphene, a protocol that uses Bloom filters and IBLTs for efficient block propagation. We have shown that Graphene is strictly more efficient than Compact Blocks unless the set of unconfirmed transactions held by peers is 1,287,670 times larger than the block size. Typically, the savings are significant on a per block basis. Additionally, using a detailed network simulation, we have demonstrated that Graphene reduces network traffic compared to the-state-of-the-art use of Compact Blocks.

References

1. Andresen, G.: O(1) Block Propagation. https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2 (August 2014)
2. Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. Commun. ACM 13(7), 422–426 (Jul 1970)
3. Corallo, M.: Bip152: Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki (April 2016)
4. Eppstein, D., Goodrich, M.T., Uyeda, F., Varghese, G.: What's the Difference?: Efficient Set Reconciliation Without Prior Context. In: ACM SIGCOMM (2011)
5. Ethereum Homestead Documentation. http://ethdocs.org/en/latest/
6. Goodrich, M., Mitzenmacher, M.: Invertible bloom lookup tables. In: Conf. on Comm., Control, and Computing. pp. 792–799 (Sept 2011)
7. Hanke, T.: A Speedup for Bitcoin Mining. http://arxiv.org/pdf/1604.00575.pdf (Rev. 5) (March 31 2016)
8. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (May 2009)
9. Russel, R.: 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)
10. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE S&P. pp. 459–474 (2014)
11. Tschipper, P.: BUIP010 Xtreme Thinblocks. https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/ (Jan 2016)