Graphene: A Probabilistic Data Structure for Efficient Propagation of Large Blocks

[DRAFT 2018-09-21]

A. Pinar Ozisik[†], Brian Levine[†], George Bissias[†], Gavin Andresen, Darren Tapp[*], Sunny Katkuri[†]
[†] *College of Computer and Information Sciences, Univ. of Massachusetts, Amherst, MA 01003*
[*]*Dash.org*

Abstract

We introduce Graphene, a method and protocol for interactive set reconciliation among peers in blockchains and related distributed systems. Through the combination of a Bloom filter and an Invertible Bloom Lookup Table (IBLT), Graphene uses a fraction of the network bandwidth used by related work for one- and two-way synchronization. We show that, for this specific problem, Graphene is $\Omega(n \log n)$ more efficient at reconciling n items than using a Bloom filter at the information theoretic bound. We contribute a fast and implementation-independent algorithm for parameterizing an IBLT so that it is optimally small in size and meets a desired decode rate with arbitrarily high probability. We characterize our performance improvements through analysis, detailed simulation, and deployment results for Bitcoin Cash, a prominent cryptocurrency. Our implementations of Graphene, IBLTs, and our IBLT optimization algorithm are all open-source code.

1 Introduction

Efficient network-based synchronization among replicas that store widely propagated information is a classic need of distributed systems. Generally, guarantees that blockchains [41,48] and protocols for distributed consensus [21,33] can scale to a large user base rely on assumptions about synchronization performance over a network. Whether based on proof-of-work [32,41], proof-of-stake [18,28], or a directed acyclic graph (DAG) [34], these systems must work aggressively to synchronize newly authored transactions and newly mined blocks of validated transactions among peers.

Synchronization among peers, or *set reconciliation*, is a critical factor in the performance of these systems. Blocks that can be relayed using less bandwidth propagate more quickly [22], thereby increasing consensus among distributed peers. When peers are unsynchronized, *forks* result from two miners producing competing blocks that have the same prior block in the chain. Forks delay consensus among miners and users alike until the fork is resolved. Systems based on GHOST [44], such as Ethereum [48], record forks on the chain forever, increasing storage bloat. Using less bandwidth to relay a block also allows greater participation by peers, who are behind limited-bandwidth links and routes (e.g., China's firewall). Finally, efficiently relaying blocks allows for the maximum block size to increase, sustaining a larger number of transactions per second overall.

Contributions. In this paper, we introduce *Graphene*, a method and protocol for synchronizing blocks (and *mempools*) among peers in blockchains and related systems using a fraction of the network bandwidth of related work. For example, for larger blocks, Graphene uses 12% of the bandwidth of existing systems. To do so, we make novel contributions to set reconciliation methods and the application of probabilistic data structures to real systems. We characterize our performance through analysis, detailed simulation, and open-source deployments. Our contributions include:

- We design a solution to the problem of efficiently stating which elements of a set M ($|M| = m$), stored by one party, are members of a set $N \subseteq M$ ($|N| = n$) selected by another party; and we apply it to a protocol for relaying a block of n transaction to a peer holding m transactions. We use a novel combination of a Bloom filter and an Invertible Bloom Lookup Table (IBLT) [30]. Our approach is smaller than using deterministic solutions [20] and previous IBLT-based approximate solutions [25]. Further, we prove that our solution to this specific problem (where both parties need the n transactions) is an improvement of $\Omega(n \log n)$ over using an optimal Bloom filter alone.
- We extend our solution to the more general case where N contains items not in M; we efficiently identify those elements and transmit them. Thus, our protocol ex-

tension handles the case where a receiver is missing transactions in the sender's block; we are a small fraction of the size of previous work [47] at the cost of an additional message. Additionally, Graphene can efficiently identify transactions held by the receiver but not the sender for general synchronization scenarios.

- We design and evaluate an efficient algorithm for parameterizing an IBLT so that it is optimally small in size but meets a desired decode rate with arbitrarily high probability and faster execution times. This result is applicable beyond our context.
- We design and evaluate a method for significantly improving the decode rate of an IBLT when two IBLTs are available. This is also a generally applicable method.
- We provide a detailed evaluation using simulations, quantifying performance against existing systems. We characterize performance of our protocol as a live Bitcoin Cash deployment, and as an Ethereum implementation for historic blocks. We also show that Graphene is more resilient to attack than previous approaches.

We have publicly released our Bitcoin Cash and Ethereum implementations of Graphene [8,12], a C++ and Python implementation of IBLTs including code for finding their optimal parameters [7], and we have released a public network specification of our basic protocol for standard interoperability [9]. Its adoption is planned by blockchain developers [4,6]. This paper improves upon our preliminary result, published previously [2].

2 Background and Related Work

Below, we summarize and contrast related work in set reconciliation and protocols for block propagation.

2.1 Set Reconciliation Data Structures

Set reconciliation protocols allow two peers, each holding a set, to obtain (and transmit) the union of the two sets. This synchronization goal is distinct from set membership protocols [19], which tell us, more simply, if an element is a member of a set. However, data structures that test set membership are useful for set reconciliation. This includes Bloom filters [14], a seminal probabilistic data structure with myriad applications [16,36,46]. Bloom filters encode membership for a set of size n by inserting the items into a small array of $\frac{-n\log_2(f)}{\ln(2)}$ bits; this efficiency gain is the result of allowing a false positive rate f.

Invertible Bloom Lookup Tables (IBLTs) [30] are a richer probabilistic data structure designed to recover the *symmetric difference* of two sets of items. Like Bloom filters, items are inserted into an IBLT's array of c *cells*, which is partitioned into subsets of size c/k. Each item is inserted once into each of the k partitions, at indices selected by k hash functions. Rather than storing only a bit, the cells store the actual item. Each cell has a *count* of the number of items inserted and the xor of all items inserted (called a *keySum*). The following algorithm [25] recovers the symmetric difference of two sets. Each set is stored in an IBLT, A and B, respectively, (with equal c and k values). For each pairwise cell of A and B, the keySums are xor'ed and the counts subtracted, resulting in a third IBLT: $A \triangle B = C$ that lacks items in the intersection. The cells in C with count$=1$ hold an item belonging to only A, and to only B if count $= -1$. These items are removed from $k-1$ other cells, which decrements their counts and allows for the additional *peeling* of new items. The process continues until all cells have a count of 0. (We've elided details about a *checkSum* field for clarity.) If c is too small given the actual symmetric difference, then iterative peeling will eventually fail, resulting in a *decode failure*, and only part of the symmetric difference will be recovered.

Comparison to Related Work. We provide a novel solution to the problem of set reconciliation, where one-way or mutual synchronization of information is required by two peers. Our results are significantly better than past work, including those based on Bloom filters alone [47] or IBLTs alone [25,30], as we show in Section 5.2.

In general, if we desire to decode sets of size j from an IBLT, we must find values $\tau > 0$ and $k > 2$, resulting in $c = j\tau$ cells (divisible by k), such that the probability of decoding is at least p. We provide an implementation-independent algorithm for finding values τ and k that meet rate p and result in the smallest value of c.

This is a significant advance over past work. Goodrich and Mitzenmacher [30] provide values of τ that *asymptotically* ensure a failure rate that decreases polynomially with j. But these asymptotic results are not optimally small in size for finite j and do not help us set the value of k optimally. Using their unreleased implementation, Eppstein et al. [25] identify optimal τ and k that meet a desired decode rate for a selection of j values; however, the statistical certainty of this optimality is unclear. In comparison, using our open-source IBLT implementation [7], we are able to systematically produce statistically optimal values τ and k for a wide range of j values. Our method, based on hypergraphs, is an order of magnitude faster than this previous method [25].

We also contribute a novel method for improving the decode rate of IBLTs, which is complementary to related work by Pontarelli et al. [42], who have the same goal.

2.2 Block Propagation

Blockchains, distributed ledgers, and related technology require a network protocol for distributing new transactions and new blocks. Almost all make use of a p2p network of peers, often a clique among *miners* that validate blocks, and a random topology among non-mining *full* nodes that store the entire chain. New transactions have an ID equal to their cryptographic hash. When a new transaction is received, a peer sends the ID as the contents of an inventory (`inv`) message to each of d neighbors, who respond with a `getdata` message if the transaction is new to them. Transactions are stored in a *mempool* until included in a valid block. Blocks are relayed similarly: an `inv` is sent to each neighbor (often the header is sent instead to save time), and a `getdata` requests the block if needed. The root of a Merkle tree [37] of all transactions validates an ordered set against the mined block.

The block consists of a header and a set of transactions. These transactions can be relayed by the sender in *full*, but this wastes bandwidth because they are probably already stored at the receiver. In other words, blocks can be relayed with a compressed encoding, and a number of schemes have been proposed. As stated in Section 1, efficient propagation of blocks is critical to achieving consensus, reducing storage bloat, overcoming network firewall bottlenecks, and allowing scaling to a large number of transactions per second.

Transactions that offer low fees to miners are sometimes marked as DoS spam and not propagated by full nodes; yet, they are sometimes included in blocks, regardless. To avoid sending redundant `inv` messages, peers keep track, on a per-transaction and per-neighbor basis, whether an `inv` has been exchanged. This log can be used by protocols to send such missing transactions to a receiver proactively as the block is relayed.

Comparison to Related Work. *Xtreme Thinblocks* [47] (XThin) is a robust and efficient protocol for relaying blocks, and is deployed in Bitcoin Unlimited (BU) clients. The receiver's `getdata` message includes a Bloom filter encoding the transaction IDs in her mempool. The sender responds with a list of the block's transaction IDs shortened to 8-bytes (since the risk of collision is still low), and uses the Bloom filter to also send any transactions that the receiver is missing. XThin's bandwidth increases with the size of the receiver's mempool, which is likely a multiple of the block size. In comparison, Graphene uses significantly lower bandwidth both when the receiver is and is not missing transactions. However, Graphene may use an additional roundtrip time to repair missing transactions.

Compact Blocks [20] is a protocol that is deployed in all Bitcoin Core and Bitcoin ABC clients. In this protocol, the receiver's `getdata` message is a simple request (no Bloom filter is sent). The sender replies with the block's transaction IDs shorted to 6-bytes (as well as the coinbase transaction). If the receiver has missing transactions, she requests repairs with a followup `inv` message. Hence, the network cost is $6n$ bytes, which is smaller than XThin's cost of $\approx \frac{m\log_2(f)}{8ln(2)} + 6n$; however, when the receiver is missing transactions, Compact Blocks has an extra roundtrip time, which may cost more if enough transactions are missing. Graphene is significantly lower in cost than Compact Blocks, as we show in Section 5.2.

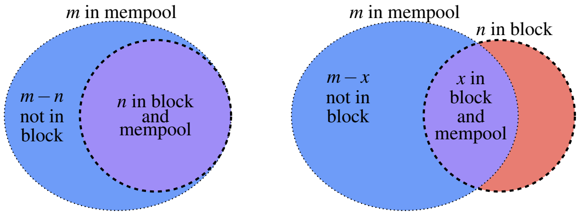

Figure 1: (Left) The receiver's mempool contains the entire block; *Protocol 1: Graphene* manages this scenario. (Right) The receiver's mempool does not contain the entire block. *Protocol 2: Graphene Extended* manages this scenario.

3 The Graphene Protocol

To motivate Graphene, consider a protocol that uses a Bloom filter alone to encode a block containing n transactions. Assume the receiver has a mempool of m transactions that are a super set of the block. If we set the FPR of the sender's Bloom filter to $f = \frac{1}{144(m-n)}$, then we can expect the filter to falsely include an extra transaction in a relayed block about once every 144 blocks (once a day in Bitcoin). This approach requires $\frac{-n\log_2(f)}{8\ln(2)}$ bytes, and it is easy to show that it is smaller than Compact Blocks ($6n$ bytes) when $m < 71982340 + n$.

But we can do better: in Graphene, we shrink the size of the Bloom filter by increasing its FPR, and we remove any false positives with an IBLT. The summed size of the two structures is smaller than using either alone. In practice, our technique performs better than Compact Blocks for any block with at least 100 transactions, and we show in Section 5.2 that it performs better than *any* Bloom-filter-only approach asymptotically.

We designed two protocols for Graphene, which we define presently. Both protocols use probabilistic data structures that fail with a tunable probability. Throughout our exposition, we use the concept of probabilistic *assurance*. Specifically, a property A is said to be held in data structure X with β-assurance whenever it is possible to tune X so that A occurs in X with probability at least β.

In **Protocol 1**, we assume that the receiver's mempool contains all transactions in the block, a typical case due to the aggressive synchronization that blockchains employ. This scenario is illustrated in Fig. 1-Left. As we show in Section 5.2, mempools are sufficiently synchronized to use only Protocol 1 about 96% of the time.

In **Protocol 2**, we do not assume that the receiver's mempool is synchronized, as illustrated in Fig. 1-Right, which allows us to apply it to two scenarios: *(i)* block relaying between unsynchronized peers; and *(ii)* intermittent mempool synchronization. A receiver may not be synchronized with the sender because of network failures, slow transaction propagation times relative to blocks, or if the block contains unpropagated low-fee transactions erroneously filtered out as spam. Protocol 2 begins when Protocol 1 fails: the receiver requests missing transactions using a second Bloom filter; and the sender transmits any missing transactions, along with a second IBLT to correct mistakes. (Compact Blocks and XThin also handle this scenario but do so with greater network bandwidth.)

3.1 Protocols

Our first protocol is for receivers whose mempool contains all the transactions in the block; see Fig. 1-Left.

PROTOCOL 1: Graphene

1: Sender: The sender transmits an `inv` (or blockheader) for a block.

2: Receiver: The receiver requests the unknown block, including a count of transactions in her mempool, m.

3: Sender: The sender creates Bloom filter $\mathbf{S}$ and IBLT $\mathbf{I}$ from the transaction IDs of the block (purple area in Fig. 1-Left). The FPR of $\mathbf{S}$ is $f_S = \frac{a}{m-n}$, and the IBLT is parameterized such that a^* items can be recovered, where $a^* > a$ with β-assurance (outlined in green in Fig. 2). We set a so as to minimize the total size of $\mathbf{S}$ and $\mathbf{I}$. $\mathbf{S}$ and $\mathbf{I}$ are sent to the receiver along with the block header (if not sent in Step 1).

4: Receiver: The receiver creates a candidate set Z of transaction IDs that pass through $\mathbf{S}$, including false positives (purple and **dark blue** areas in Fig. 2). The receiver also creates IBLT $\mathbf{I}'$ from Z. She *subtracts* $\mathbf{I} \triangle \mathbf{I}'$, which evaluates to the symmetric difference of the two sets [25]. Based on the result, she adjusts the candidate set, validates the Merkle root in the block header, and the protocol concludes.

The sender may already know the transactions for which no `inv` message has been exchanged with the receiver (e.g., Bitcoin's `filterInventoryKnown` data structure); those transactions could be sent at Step 3. N.b., the IBLT stores only 8 bytes of each transaction ID; but full IDs are used for the Bloom filter.

We use a fast algorithm to select a such that the total amount of data transmitted over the network is optimally small; see Section 3.3.1. We use a^* instead of a to parameterize $\mathbf{I}$ because Bloom filters and IBLTs are probabilistic data structures. The count of false positives from $\mathbf{S}$ has an expected mean of $(m-x)f_S = a$, whose variance comes from a Binomial distribution with parameters $(m-x)$ and f_S. We derive a^* in Section 3.3.1 via a Chernoff bound.

3.2 Graphene Extended

If the receiver does not have all the transactions in the block (Fig.1-Right), IBLT subtraction in Protocol 1 will not decode. In that case, the receiver should continue with the following protocol. Subsequently, we show how this protocol can also be used for intermittent mempool synchronization. Our contribution is not only the design of this efficient protocol, but the derivation of parameters that meet a desired decode rate.

PROTOCOL 2: Graphene Extended

1: Receiver: The size of the candidate set is $|Z| = z$, where $z = x + y$, a sum of x true positives and y false positives (purple and **dark blue** areas in Fig. 3). Because the values of x and y are obfuscated within the sum, the receiver calculates x^* such that $x^* \leq x$ with β-assurance (green outline in Fig. 3) She also calculates y^* such that $y^* \geq y$ with β-assurance (green outline in Fig. 4).

2: Receiver: The receiver creates Bloom filter $\mathbf{R}$ and adds all transaction IDs in Z to $\mathbf{R}$. The FPR of the filter is $f_R = \frac{b}{n-x^*}$, where b minimizes the size of $\mathbf{R}$ and IBLT $\mathbf{J}$ in the next step. She sends $\mathbf{R}$ and b.

3: Sender: The sender passes all transaction IDs in the block through $\mathbf{R}$. She sends all transactions that are not in $\mathbf{R}$ directly to the receiver (**red** area of Fig. 4)

4: Sender: The sender creates and sends an IBLT $\mathbf{J}$ of all transactions in the block such that $b + y^*$ items can be recovered from it. This size accounts for b, the number of transactions that falsely appear to be in $\mathbf{R}$, and y^*, the number of transactions that falsely appear to be in $\mathbf{S}$.

5: Receiver: The receiver creates IBLT $\mathbf{J}'$ from the transaction IDs in Z. She decodes the *subtraction* of the two blocks, $\mathbf{J} \triangle \mathbf{J}'$. From the result, she adjusts set Z, validates the Merkle root, and the protocol concludes.

As in Protocol 1, we set b so that the summed size of $\mathbf{R}$ and $\mathbf{J}$ is optimally small; see Section 3.3.1. We derive closed-form solutions for x^* and y^*; see Section 3.3.2.

3.2.1 Mempool Synchronization

With a few changes, Protocols 1 and 2 can be used by two peers to synchronize their mempools. Instead of a block, the sender places his entire mempool in $\mathbf{S}$ and $\mathbf{I}$. The receiver passes her mempool through $\mathbf{S}$, adding any negatives to H, the set of transactions that are not in $\mathbf{S}$. Some transactions that the sender does not have in his

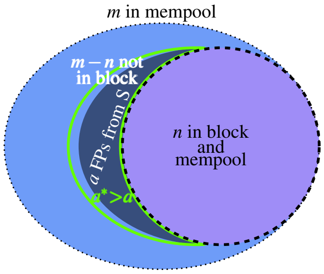

Figure 2: [Protocol 1] Passing m mempool transactions through **S** results in a FPs (in **dark blue**). A green outline illustrates $a^* > a$ with β-assurance, ensuring IBLT **I** decodes.

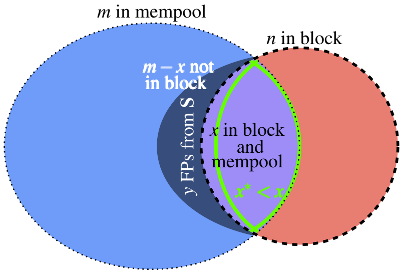

Figure 3: [Protocol 2] Passing m transactions through **S** results in z positives, obscuring a count of x TPs (**purple**) and y FPs (in **dark blue**). From z, we derive $x^* < x$ with β-assurance (in in green).

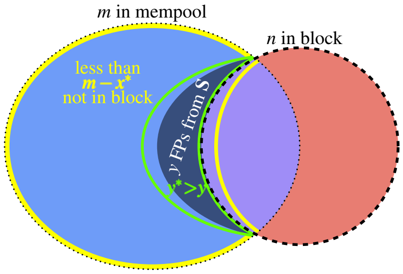

Figure 4: [Protocol 2] From our bound $m-x^* > m-x$ with β-assurance (in yellow), we can derive a bound for the false positives from **S** as $y^* > y$ with β-assurance outlined in green.

mempool will falsely pass through **S**, and these are identified by **I** (assuming that it decodes); these transactions are also added to H. If **I** does not decode, Protocol 2 is executed to find transactions in the symmetric difference of the mempools; all missing transactions among the sender and receiver are exchanged, including those in set H. The protocol is more efficient if the peer with the smaller mempool acts as the sender since **S** will be smaller. Section 5.2.2 shows the protocol is very efficient.

3.3 Ensuring Probabilistic Data Structure Success

Cryptocurrencies allow no room for error: the header's Merkle root can be validated with an exact set of transactions only. Yet, Graphene is a probabilistic solution, and if its failure rate is high, resources are wasted on recovery. In this section, we derive the parameters for Graphene that ensure a tunable, very high success rate.

3.3.1 Parameterizing Bloom filter S and IBLT I

Graphene sends the least amount of data over the network when the sum of the Bloom filter **S** and IBLT **I** is minimal. Let $T = T_{BF} + T_I$ be the summed size of the Bloom filter and IBLT. The size of a Bloom filter in bytes, T_{BF}, with false positive rate f_S and n items inserted is $T_{BF} = \frac{-n\ln(f_S)}{8\ln^2 2}$ [14]. Recall that we recover up to a^* items from the IBLT, where $a^* > a$ with β-assurance. As we show in Section 3.3.1, $a^* = (1+\delta)a$, where δ is parameterized by β. An IBLT's size is a product of the number of items recovered from a symmetric difference and a multiplier τ that ensures recovery at a desired success rate. Therefore, given the cost of r bytes per cell, T_I is

$$T_I = r\tau(1+\delta)a. \tag{1}$$

When we set $f_S = \frac{a}{m-n}$, then the total size of the Bloom filter and IBLT in bytes is

$$T(a) = \frac{-n\ln(\frac{a}{m-n})}{8\ln^2 2} + r\tau(1+\delta)a. \tag{2}$$

The value of a that minimizes T is either: $a = 1$; $a = m-n$; or the value of a where the derivative of Eq. 2 with respect to a is equal to zero, which is

$$a \approx n/(8r\tau\ln^2 2). \tag{3}$$

Eq. 3 is approximate as δ is a function of a rather than a constant. The exact value is closed form but we omit it for clarity. Furthermore, implementations of Bloom filters and IBLTs involve non-continuous ceiling functions. As a result, Eq. 3 is accurate only for $a >= 100$; otherwise the critical point a' produced by Eq. 3 can be inaccurate enough that $T(a')$ is as much as 20% higher than its true minimum value. Graphene exceeds the performance of previous work when Eq. 3 is used to select a. However, implementations that desire strictly optimal performance should take an extra step. If Eq. 3 results in a value of a less than 100, its size should be computed using accurate ceiling functions and compared against all points $a < 100$.

Derivation of a^*. We cannot parameterize IBLT **I** based on the expected number of false positives from **S** and expect a high decode rate; we must account for the natural variance of false positives generated by **S**. Here we derive a closed-form expression for a^* as a function of a and β such that $a^* > a$ holds with β-assurance, i.e. $a^* > a$ with probability at least β. Define $A_1, \ldots, A_{m-n}$ to be independent Bernoulli trials such that $Pr[A_i = 1] = f_S$, $A = \sum_{i=1}^{m-n} A_i$, and $\mu = E[A]$.

> **THEOREM 1:** *Let m be the size of a mempool that contains all n transactions from a block. If a is the actual number of false positives that result from passing the mempool through Bloom filter **S** with*

FPR f_S, then $a^* \geq a$ with probability β when

$$a^* = (1+\delta)a,$$

$$\textit{where } \delta = \frac{1}{2}(s+\sqrt{s^2+8s}) \textit{ and } s = \frac{-\ln(1-\beta)}{a}. \quad (4)$$

A full proof appears in Appendix A. According to Theorem 1, if the sender sends a Bloom filter with FPR $f_S = \frac{a}{m-n}$, then with β-assurance, no more than a^* false positives will be generated by passing elements from Z though **S**. To compensate for the variance in false positives, IBLT **I** is parametrized by a symmetric difference of $a^* = (1+\delta)a$ items. It will decode subject to its own error rate (see Section 4), provided that $a < a^*$ (which occurs with probability β) and the receiver has all n transactions in the block. We evaluate this approach in Section 5.2; see Fig. 10.

3.3.2 Parameterizing Bloom filter R and IBLT J

Parameterizing *b*. In Protocol 2, we select b so that the summed size of **R** and **J** is optimally small. It's derivation is similar to a. We show below that $y^* = (1+\delta)y$. Thus:

$$T_2(b) = \frac{z\ln(\frac{b}{n-x^*})}{8\ln^2 2} + r\tau(1+\delta)b. \quad (5)$$

The optimal value of b assuming continuous values is

$$b \approx z/(8r\tau\ln^2 2). \quad (6)$$

Similar to Section 3.3.1, an exact closed form of b exists and we omit it for clarity; and a perfectly optimal implementation would compute $T_2(b)$ using ceiling functions for values of $b < 100$.

Using z to parameterize R and J. Here we offer a closed-form solution to the problem of parameterizing **R** and **J**. This is a more challenging problem because x and y cannot be observed directly.

Let z be the observed count of transactions that pass through Bloom filter **S**. We know that $z = x+y$: the sum of x true positives and y false positives, illustrated as purple and **dark blue** areas respectively in Fig. 3. Even though x is unobservable, we can calculate a lower bound x^*, depending on x, z, m, f_S and β, such that $x^* \leq x$ with β-assurance, illustrated as a green outline in Fig. 3.

With x^* in hand, we also have, with β-assurance, an upper bound on the number of transactions the receiver is missing: $n - x^* > n - x$. This bound allows us to conservatively set $f_R = \frac{b}{n-x^*}$ for Bloom filter **R**. In other words, since $x^* < x$ with β-assurance, the sender, using **R**, will fail to send no more than b of the $n-x$ transactions actually missing at the receiver. IBLT **J** repairs these b failures, subject to its own error rate (see Section 4).

We also use x^* to calculate, with β-assurance, an upper bound $y^* \geq y$ on the number of false positives that pass through **S**. The green area in Fig. 4 shows y^*, which is a superset of the actual value for y, the **dark blue** area.

The sender's IBLT **J** contains all transactions in the block. The receiver's IBLT **J**$'$ contains true positives from **S**, false positives from **S**, and newly sent transactions. Therefore, we bound both components of the symmetric difference by $b + y^*$ transactions in order for the subtraction operation to decode. In other words, both **J** and **J**$'$ are parameterized to account for more items than actually exist in the symmetric difference between the two IBLTs.

The following theorems prove values for x^* and y^*.

THEOREM 2: *Let m be the size of a mempool containing $0 \leq x \leq n$ transactions from a block. Let $z = x+y$ be the count of mempool transactions that pass through **S** with FPR f_S, with true positive count x and false positive count y. Then $x^* \leq x$ with probability β when*

$$x^* = \arg\min_{x^*} \; Pr[x \leq x^*; z, m, f_S] \leq 1-\beta.$$

$$\textit{where } \; Pr[x \leq k; z, m, f_S] \leq \sum_{i=0}^{k} \left(\frac{e^{\delta_k}}{(1+\delta_k)^{1+\delta_k}} \right)^{(m-k)f_S}$$

$$\textit{and } \delta_k = \frac{z-k}{(m-k)f_S} - 1. \quad (7)$$

A full proof appears in Appendix A.

THEOREM 3: *Let m be the size of a mempool containing $0 \leq x \leq n$ transactions from a block. Let $z = x+y$ be the count of mempool transactions that pass through **S** with FPR f_S, with true positive count x and false positive count y. Then $y^* \geq y$ with probability β when*

$$y^* = (1+\delta)(m-x^*)f_S,$$

$$\textit{where } \delta = \frac{1}{2}(s+\sqrt{s^2+8s}) \textit{ and } s = \frac{-\ln(1-\beta)}{(m-x^*)f_S}. \quad (8)$$

A full proof appears in Appendix A.

Special case: *m* ≈ *n*.When $m \approx n$, our optimization procedure in Protocol 1 will parameterize **S** and f_S to a value near 1, which is very efficient if the receiver has all of the block. Unfortunately, when $m \approx n$ and the receiver is missing some portion of the block, Protocol 1 will fail. With $z \approx m$, Protocol 2 will set $y^* \approx m$ and $x^* \approx 0$, and $f_R \approx 1$. Most importantly, IBLT **J** will be sized to m, making it larger than a regular block.

Fortunately, resolution is straightforward. If Protocol 1 fails, and the receiver finds that $z \approx m$, $y^* \approx m$, and $f_R \approx 1$, then in Step 2 of Protocol 2, the receiver should set f_R to a fixed value. We set $f_R = 0.1$, but a large range of values execute efficiently (we tested from 0.001 to 0.2). All mempool transactions are inserted into Bloom filter **R**, and **R** is transmitted to the sender.

The sender follows the protocol as usual, sending IBLT **J** along with h transactions from the block not in **R**. However, he deviates from the protocol by also sending a third Bloom filter **F** intended to compensate for false positives from **R**. The roles of Protocol 2 are thus reversed: the sender uses Theorems 2 and 3 to solve for x^* and y^*, respectively, to bound false positives from **R** (substituting the block size for mempool size and f_R as the FPR). He then solves for b such that the total size in bytes is minimized for **F** with FPR $f_F = \frac{b}{m-h}$ and **J** having size $b+y^*$. This case may be common when Graphene is used for mempool synchronization; our evaluations in Fig. 13 in Section 5.2.2 show that our method is more efficient than Compact Blocks.

Alternatives to Bloom filters. There are dozens of variations of Bloom filters [36,46], including Cuckoo Filters [26] and Golomb Code sets [29]. To use these alternatives, Eqs. 2, 3, 5, and 6 need to be updated.

4 Enhancing IBLT Performance

The success and performance of Graphene rests heavily on IBLT performance. IBLTs have been studied in only a handful of papers [15,25,30,38,42], and current results are generally asymptotic with the size of the IBLT (the notable exception is Eppstein et al. [25], which we discuss in Section 2). In this section, we contribute several important results that allow for IBLTs to be used in practical systems with reliable precision. IBLTs are deceptively challenging to parameterize so that j items can be recovered with a desired success probability of p, using the minimal number of cells. Only two parameters can be set: the *hedge* factor, τ (resulting in $c = j\tau$ cells total), and the number of hash functions, k, used to insert an item (each function ranges over c/k cells).

Motivation. Fig. 5 motivates our contributions, showing the poor decode rate of an IBLT if static values for k and τ are applied to small values of j. The figure shows three desired decode failure rates $(1-p)$ in magenta: 1/24, 1/240, and 1/2400. The black points show the decode failure probability we observed in our IBLT implementation for static settings of $\tau = 1.5$ and $k = 4$. The resulting decode rate is either too small from a under-allocated IBLT, or exceeds the rate through over-allocation. The colored points show the failure rates of actual IBLTs parameterized by the algorithm we define below: they are optimally small and always meet or exceed the desired decode rate.

4.1 Optimal Size and Desired Decode Rate

Past work has never defined an algorithm for determining size-optimal IBLT parameters. We define an implementation-independent algorithm, adopting Malloy's [40] and Goodrich and Mitzenmacher's interpretation [30] of IBLTs as examples of uniform hypergraphs.

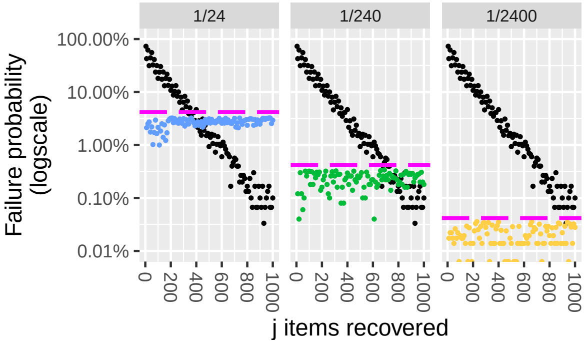

Figure 5: Parameterizing an IBLT statically results in poor decode rates. The black points show the decode failure rate for IBLTs when $k = 4$ and $\tau = 1.5$. The red, green, and blue points show decode failure rates of optimal IBLTs, which always meet a desired failure rate on each facet (in magenta). Size shown in Fig. 7.

Let $H = (V, X, k)$ be a *k-partite, k-uniform hypergraph*, composed of c vertices $V = V_1 \cup \ldots \cup V_k$ and j hyper-edges X, each connecting k vertices, one from each of the V_i. The hypergraph represents an IBLT with k hash functions, j inserted items, and c cells. Each cell corresponds to a vertex such that $|V| = c$ and $|V_i| = c/k$ (we enforce that c is divisible by k). Each item represents an edge connecting k vertices, with the ith vertex being chosen uniformly at random from V_i. Vertices V_i represent hash function i, which operates over a distinct range of cells. The *r-core* [43] of H is the maximal subgraph in which all vertices have degree at least r. H contains a non-empty 2-core iff the IBLT it represents cannot be decoded.

We seek an algorithm for determining the most space-efficient choice for c and k that is sufficient to ensure a decode rate of p for a fixed number of inserted items j. Items are inserted pseudo-randomly by applying the hash functions. Therefore, it makes sense to model the edge set X as a random variable. Define $\mathscr{H}_{j,p} = \{(V, X, k) \mid E[\texttt{decode}((V, X, k))] \geq p, |X| = j\}$, or the set of hypergraphs (V, X, k) on j edges whose expected decode success rate is bounded by p. Based on this definition, the algorithm should return

$$\underset{(V,X,k)\in\mathscr{H}_{j,p}}{\arg\min} \; |V|. \qquad (9)$$

Our approach for solving Eq. 9 is to fix j, p, and k and perform binary search over all possible values for $c = |V|$. Binary search is justified by the fact that the expected decode failure rate is a monotonically increasing function of c, which can explained as follows. A 2-core forms in (V, X, k) when there exists some group of v vertices that exclusively share a set of at least $2v$ edges. Define vertex set U such that $|U| > |V|$. Since the j edges X are chosen

ALGORITHM 1: IBLT-Param-Search

```
SEARCH($j$, $k$, $p$):
  $c_l = 1$
  $c_h = c_{max}$
  $trials == 0$
  $L = (1-p)/5$
  WHILE $c_l \neq c_h$:
      $trials += 1$
      $c = (c_l + c_h)/2$
      IF decode($j$, $k$, $c$):
          $success += 1$
      conf=conf_int($success$, $trials$)
      $r = success/trials$
      IF $r -$ conf $\geq p$:
          $c_h = c$
      IF ($r +$ conf $\leq p$):
          $c_l = c$
      IF ($r -$ conf $> p - L$) and ($r +$ conf $< p + L$):
          $c_l = c$
  RETURN $c_h$
```

Figure 6: This algorithm finds the optimally small size of $c = j\tau$ cells that decodes j items with decode success probability p (within appropriate confidence intervals) from an IBLT with k hash functions. `decode` operates over a hypergraph rather than a real IBLT.

uniformly at random, and there are more possible edges on vertex set U, the probability that a given set of $2v$ edges forms in (U, X, k) must be lower than in (V, X, k).

Fig. 6 shows the pseudocode for our algorithm, which relies on two functions. The function `decode(j,k,c)` takes a random sample from the set of hypergraphs $\mathscr{H}_{j,p}$ and determines if it forms a 2-core (i.e., if it decodes), returning `True` or `False`. The function `conf_int(s,t)` returns the 2-sided confidence interval of a proportion of s successes and t trials. In practice, we call Alg. 1 only on values of k that we have observed to be reasonable (e.g., 3 to 15), and prune the search of each k when it is clear that it will not be smaller in size than a known result.

We have released an open-source implementation of IBLTs in C++ with a Python wrapper [7]. The release includes an implementation of Alg. 1 and optimal parameters for several decode rates. Compared to a version of our algorithm that uses actual IBLTs, our hypergraph approach executes much faster for all j. For example, to parameterize $j = 100$, our approach completes in 29 seconds on average. Allocating actual IBLTs with the same code increases average run time to 426 seconds.

Fig. 7 shows the size of IBLTs when parameterized optimally for three different decode rates. If parameterized correctly, the number of cells in an IBLT grows linearly, with variations due to inherent discretization and fewer degrees of freedom in small IBLTs.

4.2 Ping-Pong Decoding

Graphene takes advantage of its two IBLTs to increase the decode rate for Protocol 2 in a novel fashion. IBLT's **I** and **J** are different sizes, and may use a different number of hash functions, but contain the same transactions. When an IBLT fails to decode completely, it still can succeed partially. The transactions that are decoded from **J** can be removed from **I**, and decoding **I** can be retried. Transactions from **I** can then be removed from **J**, and decoding **J** can be retried; and so on in a *ping-pong* fashion. We note that if the count of a decoded item is 1, then it should be subtracted from the other IBLT; if the count is -1, then it should be added to the other IBLT. The IBLTs should use different seeds in their hash functions for independence.

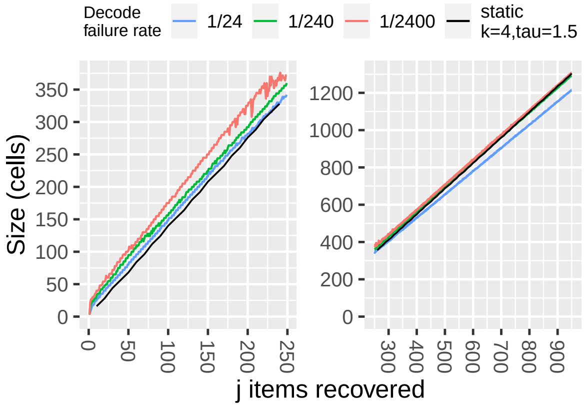

Figure 7: Size of optimal IBLTs (using Alg. 1) given a desired decode rate; with a statically parameterized IBLT ($k = 4, \tau = 1.5$) in black. For clarity, the plot is split on the x-axis. Decode rates are shown in Fig. 5.

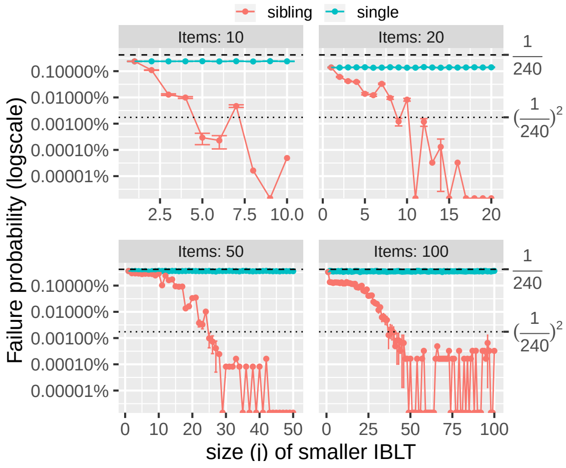

Figure 8: Decode rate of a single IBLT (parameterized for a $1/240$ failure rate) versus the improved *ping-pong decode* rate from using a second, smaller IBLT with the same items.

Fig. 8 shows an experiment where we compared the decode rate of a single IBLT parameterized to be optimally small and recover $j \in [10, 20, 50, 100]$ items with decode failure rate of $1 - p = {}^1\!/_{240}$. We then inserted the same items into a second IBLT parameterized to hold

$0 < i \leq j$ items. When i is the same size as j, the failure rate is $(1-p)^2$ or lower. But improvements can be seen for values $i < j$ as well. When j is small, very small values of i improve the decode rate. For larger values of j, larger values of i are needed for decoding. The use of ping-pong decoding on Graphene is an improvement of several orders of magnitude; results are presented in Fig. 11.

This approach can be extended to other scenarios that we do not investigate here. For example, a receiver could ask many neighbors for the same block and the IBLTs can be jointly decoded with this approach.

5 Evaluation

Our evaluation reaches the following conclusions:

- Graphene Protocol 1 is more efficient than using a Bloom filter alone, by $\Omega(n \log n)$ bits. For all but small n, it is more efficient than deterministic solutions.
- Using extensive Monte Carlo simulations, we show that Graphene Protocols 1 and 2 are always significantly smaller than Compact Blocks and XThin for a variety of scenarios, including mempool synchronization.
- In simulation, the decode success rates of Graphene Protocols 1 and 2 are above targeted values.
- We deployed Protocol 1 in Bitcoin Cash and show it performs as expected, and our implementation of Protocol 1 for Ethereum evaluated against historic data also shows expected gains.

5.1 Comparison to Bloom filter alone

The information-theoretic bound on the number of bits required to exactly describe an arbitrary unordered subset of n elements, chosen from a set of m elements is $\lceil \log_2 \binom{m}{n} \rceil \approx n \log_2(m/n)$ bits [17]. Carter et al. also showed that an approximate solution to the problem has a more efficient lower bound of $-n \log_2(f)$ bits by allowing for a false positive rate f [19].

Because our goal is to address a restricted version of this problem, Graphene Protocol 1 is more efficient than Carter's bound for even an optimal Bloom filter alone. This is because Graphene Protocol 1 assumes all n elements (transactions) are stored at the receiver, and makes use of that information whereas a Bloom filter would not.

> **THEOREM 4:** *Relaying a block with n transactions to a receiver with a mempool (a superset of the block) of m transactions is more efficient with Graphene Protocol 1 than using an optimally small Bloom filter alone, when the IBLT uses $k \geq 3$ hash functions. The efficiency gains of Graphene Protocol 1 are $\Omega(n \log_2 n)$.*

A full proof appears in Appendix B. Graphene cannot replace all uses of Bloom filters, only those where the elements are stored at the receiver, e.g., set reconciliation.

As $m - n$ approaches zero, Protocol 1 shrinks its Bloom filter and approaches an IBLT-only solution. If we check the special case of Graphene having an FPR of 1 (equivalent to not sending a Bloom filter at all) then Graphene is as small as any IBLT-only solution, as expected; As $m - n$ increases, Graphene is much smaller than sticking with an IBLT-only solution, which would have $\tau(m-n)$ cells.

Graphene is not always smaller than deterministic solutions. As we show in our evaluations below, for small values of n (about 100 or fewer), deterministic solutions perform better. For larger values, Graphene's savings are significant and increase with n.

We leave analytic claims regarding Protocol 2 for future work; however, below we empirically demonstrate its advantage over related work.

5.2 Monte Carlo Simulation

Our comparisons are against Compact Blocks [20] because it performs well and is deployed most widely. It does not use any probabilistic data structures. We did not compare against XThin because it is higher bandwidth than Compact Blocks and Graphene; as noted in Section 2, it has the advantage over both approaches of one less roundtrip of exchanged messages.

Methodology and Assumptions. We wrote a custom block propagation simulator for Graphene that measures the network bytes exchanged by peers relaying blocks. We executed the protocol using real data structures so that we could capture the probabilistic nature of Bloom filters and IBLTs. Specifically, we used our publicly released IBLT implementation and a well-known Python Bloom filter implementation. In results below, we varied several key parameters, including the size of the block, the size of the receiver's mempool, and the fraction of the block possessed at the receiver. Each point in our plots is one parameter combination and shows the mean of 10,000 trials or more; if no confidence interval is shown, it was extremely small and removed for clarity. For all trials, we used a bound of $\beta = 239/240$ (see Eqs. 18 and 30).

In all experiments, we evaluated three block sizes (in terms of transactions): 200, which is about the average size of Ethereum (ETH) and Bitcoin Cash (BCH) blocks; 2,000 which is the average size of Bitcoin (BTC) blocks; and 10,000 as an example of a larger block scenario. In expectation of being applied to large blocks and mempools, we used 8-byte transaction IDs for both Graphene and Compact Blocks. Also for Compact Blocks, we use `getdata` messages with block encodings of 1 or 3 bytes, depending on block size [20].

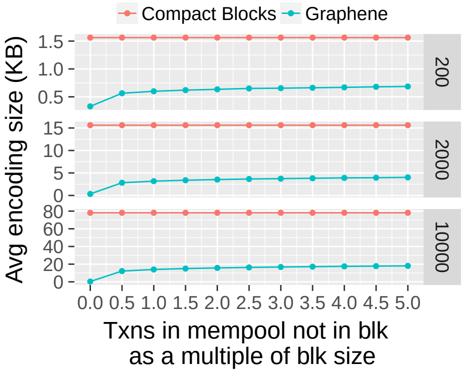

Figure 9: [Simulation, Protocol 1] Average size of Graphene blocks versus Compact Blocks as the size of the mempool increases as a multiple of block size. Each facet is a block size: (200, 2000, and 10000 transactions).

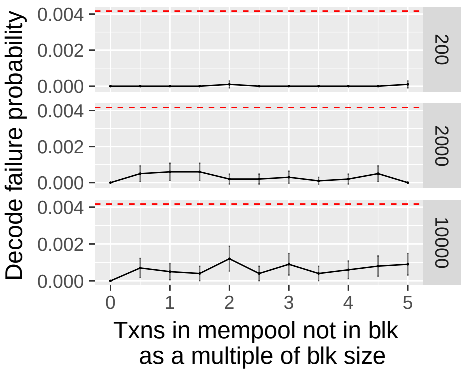

Figure 10: [Simulation, Protocol 1] Decode rate of Graphene blocks with a Chernoff bound of $\beta = \frac{239}{240}$ (red dotted line), as block size and the number of extra transactions in the mempool increases as a multiple of block size.

5.2.1 Graphene: Protocol 1

Size of blocks. Fig. 9 shows the cost in bytes of Graphene blocks compared to Compact Blocks. In these experiments, the receiver's mempool contains all transactions in the block plus some additional transactions, which increase along the x-axis as a multiple of the block size. For example, at fraction 0.5 and block size 2,000, the mempool contains 3,000 transactions in total. The experiments demonstrate that Graphene's advantage over Compact Blocks is substantial and improves with block size. Furthermore, the cost of Graphene grows sublinearly as the number of extra transactions in the mempool grows.

Decode rate. Fig. 10 shows the decode rate of Graphene blocks, as the mempool size increases. In all cases, the decode rate far exceeds the desired rate, demonstrating that our derived bounds are very effective. Graphene's decode rate suffers when the receiver lacks the entire block in her mempool. For example, in our experiments, a receiver holding 99% of the block can still decode 97% of the time. But if the receiver holds less than 98% of the block, the decode rate for Protocol 1 is zero. Hence, Protocol 2 is required in such scenarios.

5.2.2 Graphene Extended: Protocol 2

Our evaluations of Protocol 2 focus on scenarios where the receiver does not possess the entire block and $m > n$; we evaluate $m = n$ as a special case.

Size by message type. Fig. 12 shows the cost of Graphene Extended, broken down into message type, as the fraction of the block owned by the receiver increases. The dashed line on the plot shows the costs for Compact Blocks, where the receiver requests missing transactions by identifying each as a 1- or 3-byte index (depending on block size) in the original ordered list of transactions in the block encodings [20]. (We exclude the cost of sending the missing transactions themselves for both protocols.)

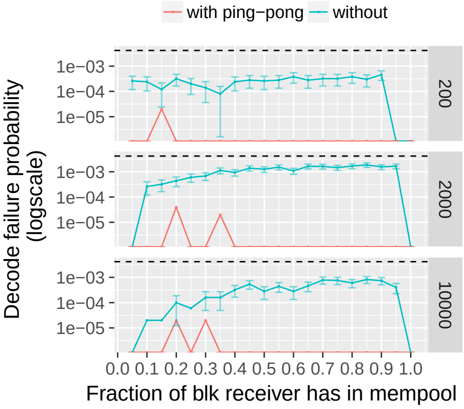

Figure 11: [Simulation, Protocol 2] Decode rate of Graphene blocks with a Chernoff bound of $\beta = \frac{239}{240}$, shown by the red dotted line, as block size and the number of extra transactions in the mempool increases. Error bars represent 95% confidence intervals.

Overall, Graphene Extended is significantly smaller than Compact Blocks, and the gains increase as the block size increases. For blocks smaller than 200, eventually Compact Blocks would be smaller in some scenarios.

Decode rate.

Fig. 11 shows the decode rate of Graphene blocks; not only does it far exceed the desired rate, but approaches very close to 100% with the use of ping-pong decoding.

Not shown are our simulations of the Difference Digest by Eppstein et al. [25], which is several times more expensive than Graphene. The Difference Digest is an IBLT-only solution that is an alternative to our Protocol 2.

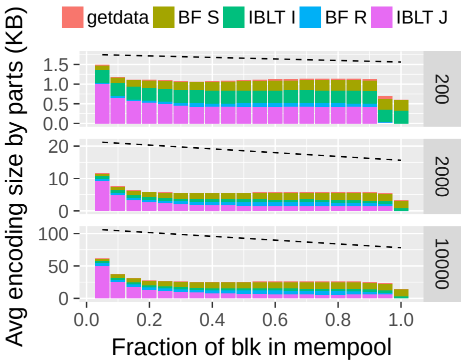

Figure 12: [Simulation, Protocol 2] Graphene Extended cost as the fraction of the block owned by the receiver increases. Black-dotted line is the cost of Compact Blocks.

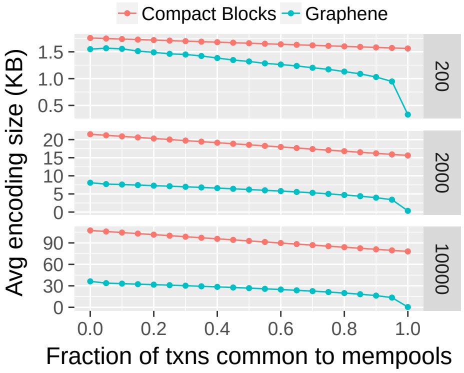

Figure 13: [Simulation, Mempool Synchronization] Here $m = n$ and the peers have a fraction of the sender's mempool in common on the x-axis. Graphene is more efficient, and the advantage increases with block and mempool size.

In that work, the sender begins by telling the receiver the value n. The receiver creates a Flajolet-Martin estimate [27] of $m-n$ using $\lceil \log_2(m-n) \rceil$ IBLTs each with 80 cells with roughly m elements inserted. The sender replies with a single IBLT of twice the number of cells as the estimate (to account for an under-estimate).

$m \approx n$ and mempool synchronization. As described in Section 3.2.1, Graphene can be used for mempool synchronization, setting n to the size of the sender's mempool. In these cases, if the peers are mostly synchronized, then $m \approx n$, which is a special case for Graphene discussed in Section 3.3.1. Our evaluations of this scenario are shown in Fig. 13. In these experiments, the sender's mempool has n transactions, of which a fraction (on the x-axis) are in common with the receiver. The receiver's mempool size is topped off with unrelated transaction so that $m = n$. As a result, Protocol 1 fails and modifications from Section 3.3.1 are employed. As with previous experiments, Graphene performs significantly better than Compact Blocks across multiple mempool intersection sizes and the improvement increases with block size.

5.3 Implementations

Bitcoin Cash Implementation. We coded Graphene (Protocol 1) for Bitcoin Unlimited's Bitcoin Cash edition 1.4.0.0, released on August 17, 2018, as an optional, experimental feature at their request. Currently, 28 nodes (operated by persons unknown to us) are running Graphene on the Bitcoin Cash mainnet. An updated count of nodes can be found at [1]. Graphene is part of the formal plans for two major clients on Bitcoin Cash (BCH): Unlimited [6] and ABC [4].

Fig. 14 shows results from our own peer running the protocol on the real network. The results show the size of Graphene encodings when there wasn't a Protocol 1 failure; we have not deployed Protocol 2 as of yet. Failures occurred roughly 4% of the time due to unsynchronized mempools between our peer and our single neighbor. Our investigations suggest that using the `filterInventoryKnown` data structure would allow the sender to predict which transactions are missing at the receiver, reducing the failure rate of Protocol 1 significantly. This statistic also confirms our two-protocol approach: 96% of the time Protocol 1 is sufficient; and correcting failures with Protocol 2 is needed 4% of the time. Fig. 14 also shows results from using Bitcoin Unlimited's XThin implementation; however, we have removed the cost of the receiver's Bloom filter to make the comparison fair (hence it is labelled *XThin**). The same plot separates the costs of providing transaction ordering information; see Section 6.2. It shows that Graphene is efficient and scales linearly with the number of transactions in the block.

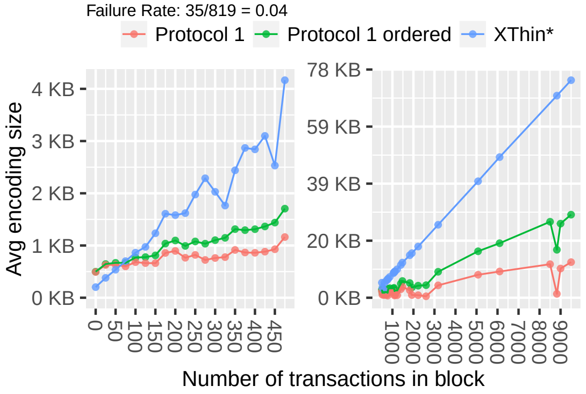

Figure 14: [Deployment on BCH, Protocol 1]: Performance of Protocol 1 as deployed on the Bitcoin Cash network. The node was connected to one other peer running the same code. The x-axis is split across two facets for clarity.

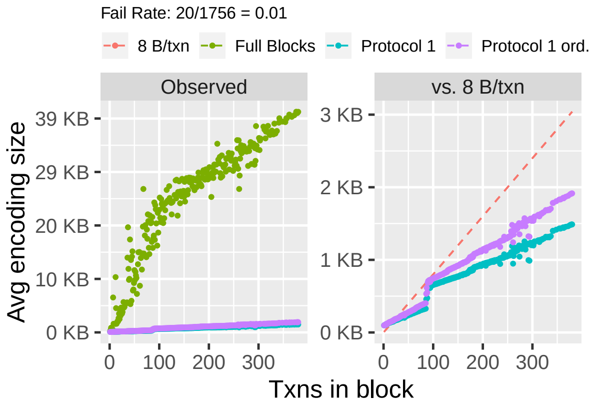

Figure 15: [Implementation, Protocol 1] An implementation of Protocol 1 for the Geth Ethereum client run on historic data. The left facet compares against Ethereum's use of full blocks; the right compares against an idealized version of Compact Blocks using 8 bytes/transaction.

Ethereum Implementation. We implemented Graphene for *Geth*, Ethereum's primary client software, and submitted a Pull Request [12]. We replayed 1,756 historic blocks with typical block sizes, introducing new message types to comply with Graphene's protocol. During our test, the size of the mempool at the receiver was kept constant at 60,000 transactions, which is typical (see `https://etherscan.io/chart/pendingtx`). The left facet of Fig. 15 shows the size in bytes of full blocks used by Ethereum and of Graphene. The right facet compares Graphene (including transaction ordering information) against a line showing 8 bytes/per transaction (an idealization of Compact Blocks without overhead).

6 Systems Issues

6.1 Security Considerations

Malformed IBLTs. It is simple to produce an IBLT that results in an endless decode loop for a naive implementation; the attack is just as easily thwarted. To create a malformed IBLT, the attacker incorrectly inserts an item into only $k-1$ cells. When the item is peeled off, one cell in the IBLT will contain the item with a count of -1. When that entry is peeled, $k-1$ cells will contain the item with a count of 1; and the loop continues. The attack is thwarted if the implementation halts decoding when an item is decoded twice. Once detected, the sender can be dropped or banned by the receiver.

Manufactured Transaction Collisions. The probability of accidental collision of two 8-byte transaction IDs in a mempool of size m is $\approx 1 - \text{Exp}\left(\frac{-m(m-1)}{2^{65}}\right)$ [39]. An attacker may use brute force search to discover and submit collisions. SipHash [13] is used by some blockchain protocols to limit the attack to a single peer.

With or without the use of SipHash, Graphene is more resilient against such collisions than XThin and Compact Blocks. Let t_1 and t_2 be transactions with IDs that collide with at least 8 bytes. In the worst case, the block contains t_1, the sender has never seen t_2, and the receiver possesses t_2 but has never seen t_1. In this case, XThin and Compact Blocks will always fail; however, Graphene fails with low probability, $f_S \cdot f_R$. For the attack to succeed, first, t_2 must pass through Bloom filter **S** as a full 32-byte ID, which occurs only with probability f_S. If it does pass, the IBLT will decode but the Merkle root will fail. At this point, the receiver will initiate Protocol 2, sending Bloom filter **R**. Second, with probability f_R, t_1 will be a false positive in **R** as a full 32-byte ID and will not be sent to the receiver.

6.2 Transaction Ordering Costs

Bloom filters and IBLTs operate on unordered sets, but Merkle trees require a specific ordering. In our evaluations, we did not include the cost for the sender in Graphene to specify the order of transactions, which is $n \log_2 n$ bits. As n grows, this cost is larger than Graphene itself. Fortunately, the cost can be removed by blockchains. A canonical ordering scheme could be enforced network-wide. Alternatively, an optional ordering, such as lexical sorting, can be specified by a miner using a flag that is passed along with the relayed block. Several developers have proposed a variety of ordering schemes to support Graphene and other features [3,5,10,11].

6.3 Reducing Processing Time

Profiling our implementation code revealed that processing costs are dominated heavily by passing the receiver's mempool against Bloom filter **S** in Protocol 1. Fortunately, this cost is easily reduced. A standard Bloom filter implementation will hash each transaction ID k times — but each ID is already the result of applying a cryptographic hash and there is no need to hash k more times; see Suisani et al. [45]. Instead, we break the 32-byte transaction ID into k pieces. Applying this solution reduced average receiver processing in our Ethereum implementation from 17.8ms to 9.5ms. Alternative techniques [23,24,31] are also effective and not limited to small values of k.

6.4 Limitations

Graphene is a solution for set reconciliation where there is a trade-off between transmission size, complexity (in terms of network round-trips), and success rate. In contrast, popular alternatives such as Compact Blocks [20] have predictable transmission size, fixed transmission complexity, and always succeed. As a result, although expected performance for each criteria is as good or better, Graphene cannot guarantee superiority. There always exists a non-zero probability that Graphene will fail in Protocol 1 or 2, requiring additional round-trips. For small set sizes ($n < 100$), Graphene could be inferior in terms of transmission size. Hence, careful consideration is necessary when deciding to deploy Graphene.

7 Conclusions

We introduced a novel solution to the problem of determining a subset of items from a larger set two parties hold in common, using a novel combination of Bloom filters and IBLTs. We also provided a solution to the more general case, where one party is missing some or all of the subset. Specifically, we described how to parametrize the probabilistic data structures in order to meet a desired decode rate. Through a detailed evaluation using simulations and real-world deployment, we compared our method to existing systems, showing that it requires less data transmission over a network and is more resilient to attack than previous approaches.

Acknowledgements. This work was funded in part by Bitcoin Unlimited, and was performed in part using high performance computing equipment obtained under a grant from the Collaborative R&D Fund managed by the Massachusetts Technology Collaborative. We are grateful for insightful feedback from Andrew Stone, Peter Tschipper, Andrea Suisani, Awemany, and Peter Rizun.

References

[1] Bitcoin Cash Nodes. https://cashnodes.io/nodes. Search for "graphene".

[2] A. Pinar Ozisik and Gavin Andresen and George Bissias and Amir Houmansadr and Brian Neil Levine. Graphene: A New Protocol for Block Propagation Using Set Reconciliation. Proc. of International Workshop on Cryptocurrencies and Blockchain Technology (ESORICS Workshop). http://forensics.cs.umass.edu/pubs/ozisik.cbt2017.pdf.

[3] Awemany. TopoCanonical ordering - and more efficient graphene transmission. https://github.com/BitcoinUnlimited/BitcoinUnlimited/pull/1275, Aug 20 2018.

[4] Bitcoin ABC. The Bitcoin ABC Vision. https://medium.com/@Bitcoin_ABC/the-bitcoin-abc-vision-f7f87755979f, Aug 24 2018.

[5] Bitcoin ABC. Benefits of Canonical Transaction Order. https://www.bitcoinabc.org/2018-08-15-benefits-of-ctor/, Aug 15 2018.

[6] Bitcoin Unlimited. Bitcoin Cash Development And Testing Accord: Bitcoin Unlimited Statement. https://www.bitcoinunlimited.info/cash-development-plan, 2018.

[7] Brian Levine and Gavin Andresen. IBLT Optimization (open-source repository). https://github.com/umass-forensics/IBLT-optimization, August 2018.

[8] George Bissias. Graphene Pull Request. https://github.com/BitcoinUnlimited/BitcoinUnlimited/pull/973, July 2018.

[9] George Bissias and Brian Levine. BUIP093: Graphene Relay. https://github.com/BitcoinUnlimited/BUIP/blob/master/093.mediawiki, July 26 2018.

[10] Shammah Chancellor. Sharding Bitcoin Cash . https://medium.com/@Bitcoin_ABC/sharding-bitcoin-cash-35d46b55ecfb, Aug 27, 2018.

[11] Stone, G. Andrew. Why ABC's CTOR Will Not Scale. https://medium.com/@g.andrew.stone/why-abcs-ctor-will-not-scale-8a6c6cf4a441, Sept 7 2018.

[12] Sunny Katkuri. Improve block transfer efficiency using Graphene #17724. https://github.com/ethereum/go-ethereum/pull/17724, Sept 20 2018.

[13] Aumasson, J.-P., and Bernstein, D. J. SipHash: A Fast Short-Input PRF. In *Progress in Cryptology (INDOCRYPT)* (2012), pp. 489–508.

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

[15] Boral, A., and Mitzenmacher, M. Multi-party set reconciliation using characteristic polynomials. In *Proc. Annual Allerton Conference on Communication, Control, and Computing* (Oct 2014).

[16] Broder, A., and Mitzenmacher, M. Network applications of bloom filters: A survey. *Internet mathematics 1*, 4 (2004), 485–509.

[17] Brodnik, A., and Munro, J. I. Membership in constant time and almost-minimum space. *SIAM Journal on Computing 28*, 5 (1999), 1627–1640.

[18] Buterin, V., and Griffith, V. Casper the friendly finality gadget. https://arxiv.org/abs/1710.09437, Oct 2017.

[19] Carter, L., Floyd, R., Gill, J., Markowsky, G., and Wegman, M. Exact and approximate membership testers. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1978), STOC '78, ACM, pp. 59–65.

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

[21] Danezis, G., and Meiklejohn, S. Centrally banked cryptocurrencies. In *Proc. Network and Distributed System Security Symposium (NDSS)* (Feb 2016).

[22] Decker, C., and Wattenhofer, R. Information Propagation in the Bitcoin Network. In *13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy* (September 2013).

[23] Dillinger, P. C., and Manolios, P. Bloom filters in probabilistic verification. In *In Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD* (2004), Springer-Verlag, pp. 367–381.

[24] Dillinger, P. C., and Manolios, P. Fast and accurate bitstate verification for spin. *Lecture Notes in Computer Science* (2004), 57–75.

[25] Eppstein, D., Goodrich, M. T., Uyeda, F., and Varghese, G. What's the Difference?: Efficient Set Reconciliation Without Prior Context. In *ACM SIGCOMM* (2011).

[26] Fan, B., Andersen, D. G., Kaminsky, M., and Mitzenmacher, M. D. Cuckoo filter: Practically better than bloom. In *Proc. ACM CoNEXT* (2014), pp. 75–88.

[27] Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31*, 2 (1985), 182 – 209.

[28] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proc. Symposium on Operating Systems Principles (SOSP)* (2017), pp. 51–68.

[29] Golomb, S. W. Run-length encodings, determining explicit form of huffman coding when applied to geometric distribution. *EEE Trans Info Theory 12*, 3 (1966), 399–401.

[30] Goodrich, M., and Mitzenmacher, M. Invertible bloom lookup tables. In *Conf. on Comm., Control, and Computing* (Sept 2011), pp. 792–799.

[31] Kirsch, A., and Mitzenmacher, M. Less hashing, same performance: Building a better bloom filter. In *Algorithms – ESA 2006* (Berlin, Heidelberg, 2006), Y. Azar and T. Erlebach, Eds., Springer Berlin Heidelberg, pp. 456–467.

[32] Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Proc. USENIX Security Symposium* (2016), pp. 279–296.

[33] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *Proc. IEEE Symposium on Security and Privacy* (May 2018), pp. 583–598.

[34] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. Inclusive block chain protocols. In *Proc. International Conference on Financial Cryptography and Data Security* (Jan 2015), pp. 528–547.

[35] Love, E. R. Some logarithm inequalities. *The Mathematical Gazette (The Mathematical Association) 63*, 427 (`https://www.jstor.org/stable/3615890` 1980), 55–57.

[36] Luo, L., Guo, D., Ma, R. T., Rottenstreich, O., and Luo, X. Optimizing bloom filter: Challenges, solutions, and comparisons. *arXiv preprint arXiv:1804.04777* (2018).

[37] Merkle, R. C. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87* (Berlin, Heidelberg, 1988), C. Pomerance, Ed., Springer Berlin Heidelberg, pp. 369–378.

[38] Mitzenmacher, M., and Pagh, R. Simple multi-party set reconciliation. *Distributed Computing* (Oct 2017).

[39] Mitzenmacher, M., and Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[40] Molloy, M. The pure literal rule threshold and cores in random hypergraphs. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2004), SODA '04, Society for Industrial and Applied Mathematics, pp. 672–681.

[41] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2009.

[42] Pontarelli, S., Reviriego, P., and Mitzenmacher, M. Improving the performance of Invertible Bloom Lookup Tables. *Information Processing Letters 114*, 4 (2014), 185 – 191.

[43] Seidman, S. B. Network structure and minimum degree. *Social Networks 5*, 3 (1983), 269 – 287.

[44] Sompolinsky, Y., and Zohar, A. Secure high-rate transaction processing in Bitcoin. *Financial Cryptography and Data Security* (2015).

[45] Suisani, A., Clifford, A., Stone, A., Beijnoff, E., Rizun, P., Tschipper, P., Fedorova, A., Feng, C., Lemieux, V., and Matthews, S. Measuring maximum sustained transaction throughput on a global network of Bitcoin nodes. In *Proc. Scaling Bitcoin* (November 2017).

[46] Tarkoma, S., Rothenberg, C. E., and Lagerspetz, E. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys Tutorials 14*, 1 (First 2012), 131–155.

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

[48] Wood, G. Ethereum: A secure decentralised generalised transaction ledger. `https://ethereum.github.io/yellowpaper/paper.pdf`, June 2018.

A Theorems from Section 3.3

For completeness, we provide the proof of a well-known version of Chernoff bounds that appears commonly in lecture notes, but not in any formal reference to our knowledge.

> **LEMMA 1:** *Let A be the sum of i independent Bernoulli trials $A_1,\ldots,A_i$, with mean $\mu = E[A]$. Then for $\delta > 0$*
>
> $$Pr[A \geq (1+\delta)\mu] \leq Exp\left(-\frac{\delta^2}{2+\delta}\mu\right), \tag{10}$$

PROOF: Starting from the well-known Chernoff bound [39]:

$$\begin{aligned} Pr[A \geq (1+\delta)\mu] &\leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} && (11)\\ &= \text{Exp}(\mu(\delta-(1+\delta)ln(1+\delta))) && (12)\\ &\leq \text{Exp}\left(\mu\left(\delta-(1+\delta)\left(\frac{2\delta}{2+\delta}\right)\right)\right) && (13)\\ &= \text{Exp}\left(\frac{-\delta^2}{2+\delta}\mu\right) && (14)\end{aligned}$$

Above, we rely on the inequality $ln(1+x) \geq \frac{x}{1+x/2} = \frac{2x}{2+x}$ for $x > 0$ (see [35]), and that $e^{a-b} \leq e^{a-c}$ when $b \geq c$. □

> **THEOREM 1:** *Let m be the size of a mempool that contains all n transactions from a block. If a is the actual number of false positives that result from passing the mempool through Bloom filter $\mathbf{S}$ with FPR f_S, then $a^* \geq a$ with probability β when*
>
> $$\begin{aligned} a^* =&(1+\delta)a,\\ \textit{where } \delta =&\frac{1}{2}(s+\sqrt{s^2+8s}) \textit{ and } s = \frac{-\ln(1-\beta)}{a}. \end{aligned} \tag{15}$$

PROOF: There are $m-n$ potential false positives that pass through $\mathbf{S}$. They are a set $A_1,\ldots,A_{m-n}$ of independent Bernoulli trials such that $Pr[A_i = 1] = f_S$. Let

$A = \sum_{i=1}^{m-n} A_i$ and $\mu = E[A] = f_S(m-n) = \frac{a}{m-n}(m-n) = a$. From Lemma 1, we have

$$Pr[A \geq (1+\delta)\mu] \leq \text{Exp}\left(-\frac{\delta^2}{2+\delta}\mu\right), \qquad (16)$$

for $\delta \geq 0$. The receiver can set a bound of choice, $0 < \beta < 1$, and solve for δ using the right hand side of Eq. 16. To bound with high probability, we seek the complement of the right hand side

$$\beta = 1 - \text{Exp}\left(-\frac{\delta^2}{2+\delta}a\right) \qquad (17)$$

$$\delta = \frac{1}{2}(s + \sqrt{s^2 + 8s}), \text{ where } s = \frac{-\ln(1-\beta)}{a}. \qquad (18)$$

□

> **THEOREM 2:** *Let m be the size of a mempool containing* $0 \leq x \leq n$ *transactions from a block. Let* $z = x + y$ *be the count of mempool transactions that pass through* **S** *with FPR* f_S*, with true positive count x and false positive count y. Then* $x^* \leq x$ *with probability* β *when*
>
> $$x^* = \underset{x^*}{\arg\min}\ Pr[x \leq x^*; z, m, f_S] \leq 1 - \beta.$$
>
> $$\textit{where}\ \ Pr[x \leq k; z, m, f_S] \leq \sum_{i=0}^{k} \left(\frac{e^{\delta_k}}{(1+\delta_k)^{1+\delta_k}}\right)^{(m-k)f_S}$$
>
> $$\textit{and}\ \delta_k = \frac{z-k}{(m-k)f_S} - 1. \qquad (19)$$

PROOF: We can't observe the values x or y, but whatever their real values, we know their dependency: $Y = \sum_{i=1}^{m-x} Y_i$, where $Y_1, \ldots, Y_{m-x}$ are independent Bernoulli trials such that $Pr[Y_i = 1] = f_S$.

For a given value x, we can compute $Pr[Y \geq y]$, the probability of at least y false positives passing through the sender's Bloom filter. We apply a Chernoff bound [39]:

$$Pr[y; z, x, m] =$$

$$Pr[Y \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \qquad (20)$$

where $\delta > 0$, and $\mu = E[Y] = (m-x)f_S$. By setting $(1+\delta)\mu = z - x$ and solving for δ, we have

$$(1+\delta)(m-x)f_S = z - x \qquad (21)$$

$$\delta = \frac{z-x}{(m-x)f_S} - 1. \qquad (22)$$

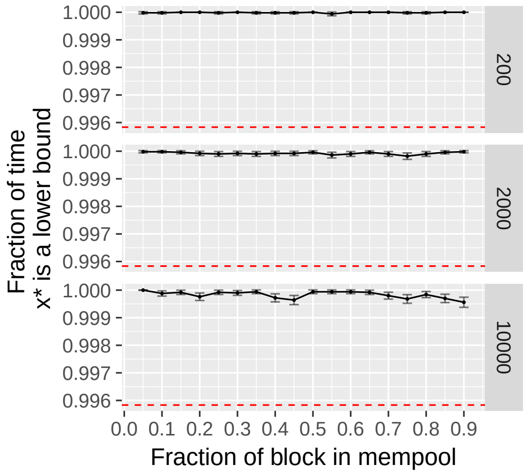

Figure 16: [Simulation] The fraction of Monte Carlo experiments where $x^* < x$ via Theorem 2 compared to a desired bound of $\beta = 239/240$ (shown as a dashed red line).

We substitute δ into Eq. 20 and bound the probability of observing a value of $y = z - x$ or greater, given that the receiver has x transactions in the block. This realization allows us to enumerate all possible scenarios for observation z. The cumulative probability of observing y, parametrized by z, given that the receiver has at most k of the transactions in the block, is:

$$Pr[x \leq k; z, m, f_S] = \sum_{i=0}^{k} Pr[y; z, k, m] \qquad (23)$$

$$\leq \sum_{i=0}^{k} \left(\frac{e^{\delta_k}}{(1+\delta_k)^{1+\delta_k}}\right)^{(m-k)f_S} \qquad (24)$$

where $\delta_k = \frac{z-k}{(m-k)f_S} - 1$. Finally, using this closed-form equation, we select a bounding probability β, such as $\beta = 239/240$. We seek a probability β of observing z from a value x^* or larger; equivalently, we solve for the complement:

$$\underset{x^*}{\arg\min}\, Pr[x \leq x^*; z, m, f_S] \leq 1 - \beta. \qquad (25)$$

To summarize, x^* is the *smallest* number of true positives such that the cumulative probability of observing $y = z - x^*$ false positives is at least $1 - \beta$.

□

For good measure, we validated the theorem empirically, as shown in Fig. 16.

> **THEOREM 3:** *Let m be the size of a mempool containing* $0 \leq x \leq n$ *transactions from a block. Let* $z = x + y$ *be the count of mempool transactions that pass through* **S** *with FPR* f_S*, with true positive count x and false positive count y. Then* $y^* \geq y$ *with prob-*

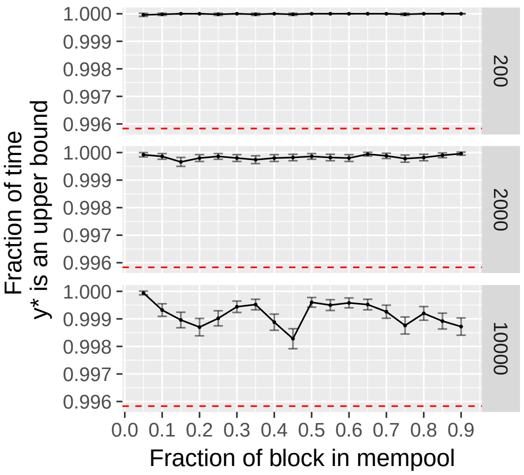

Figure 17: [Simulation, Protocol 2] The fraction of Monte Carlo experiments where $y^* > y$ via Theorem 3 compared to a desired bound of $\beta = 239/240$ (shown as a dashed red line).

> *ability β when*
>
> $$y^* = (1+\delta)(m-x^*)f_S,$$
>
> $$\textit{where } \delta = \frac{1}{2}(s+\sqrt{s^2+8s}) \textit{ and } s = \frac{-\ln(1-\beta)}{(m-x^*)f_S}. \tag{26}$$

PROOF: First, we solve for $x^* \leq x$ with β-assurance using Theorem 2. We find $y^* = z - x^* \geq y$ by applying Lemma 1 to $Y = \sum_{i=1}^{m-x^*}$, the sum of $m - x^*$ independent Bernoulli trials such that $Pr[Y_i = 1] = f_S$ trials and $\mu = (m-x^*)f_S$:

$$Pr[Y \geq (1+\delta)\mu] \leq \text{Exp}\left(-\frac{\delta^2}{2+\delta}\mu\right), \tag{27}$$

for $\delta \geq 0$. We select $0 < \beta < 1$, and solve for δ using the right hand side of Eq. 27. To bound with high probability, we seek the complement of the right hand side.

$$\beta = 1 - \text{Exp}\left(-\frac{\delta^2}{2+\delta}(m-x^*)f_S\right) \tag{28}$$

$$\tag{29}$$

$$\delta = \frac{1}{2}(s+\sqrt{s^2+8s}), \text{ where } s = \frac{-\ln(1-\beta)}{(m-x^*)f_S} \tag{30}$$

Then, we set

$$y^* = (1+\delta)(m-x^*)f_S. \tag{31}$$

Since, $x^* \leq x$ with β-assurance, it follows that y^* also bounds the sum of $m - x$ Bernoulli trials, where

$$y^* = (1+\delta)(m-x)f_S, \tag{32}$$

with probability at least β for any $\delta \geq 0$ and $m > 0$. □

We validated this theorem empirically as well, as shown in Fig. 17.

B Theorems from Section 5.1

> **THEOREM 4:** *Relaying a block with n transactions to a receiver with a mempool (a superset of the block) of m transactions is more efficient with Graphene Protocol 1 than using an optimally small Bloom filter alone, when the IBLT uses $k \geq 3$ hash functions. The efficiency gains of Graphene Protocol 1 are $\Omega(n \log_2 n)$.*

PROOF: We assume that $m = cn$ for some constant $c > 1$. Our proof is asymptotic. Thus, according to the law of large numbers, every value $\delta > 0$ (where δ is defined as in Theorem 1) is sufficient to achieve β-assurance when choosing values for a^*, x^*, and y^*. Accordingly, we may proceed under the assumption that $\delta = 0$, i.e. there is no need to hedge the false positive rate of either Bloom filter lower to account for deviations because the observed false positive rate will always match its expected value asymptotically.

Let f, where $0 < f < 1$, be the FPR of a Bloom filter created in order to correctly identify $n \geq 1$ elements from a set of $m \geq 1$ elements. The size of the Bloom filter that has FPR, f, with n items inserted, is $-n\log_2(f)$ bits [19]. Let $f = \frac{p}{m-n}$, where $0 < p < 1$. The expected number of false positives that can pass through the Bloom filter is $(m-n)\frac{p}{(m-n)} = p$. Since $0 < p < 1$, one out of every $1/p$ Bloom filters is expected to fail.

To correctly identify the same set of items, Graphene instead uses a Bloom filter with $f = \frac{a}{m-n}$, where we set $a = n/rt$ since the Bloom filter is optimal, and use an IBLT with $a\tau$ cells (r bytes each) that decodes with probability p. The expected number of false positives that pass through Graphene's Bloom filter is $(m-n)\frac{a}{(m-n)} = a$. An IBLT with 1 to a items inserted in it decodes with probability $1 - p$. In other words, one out of every $1/p$ Graphene blocks is expected to fail.

The difference in size is

$$-n\log_2\left(\frac{p}{m-n}\right) - \left(-n\log_2\left(\frac{a}{m-n}\right) + ar\tau\right) \tag{33}$$

$$= n\log_2(a/p) - ar\tau \tag{34}$$

$$= n(\log_2 n + \log_2 {}^1\!/_{p\tau}) - 1) \tag{35}$$

$$= n(\log_2 n + \Omega(\tau^{2-k})) \tag{36}$$

$$= \Omega(n(\log_2 n)), \tag{37}$$

where Eq. 36 follows from Theorem 1 from Goodrich and Mitzenmacher [30], given that we have an IBLT with $k \geq 3$ hash functions. □