Space- and Computationally-Efficient Set Reconciliation via Parity Bitmap Sketch (PBS)

Long Gong[†] Ziheng Liu[††] Liang Liu[†] Jun Xu[†] Mitsunori Ogihara[‡] Tong Yang[††]
[†]Georgia Institute of Technology, Atlanta, USA
{gonglong,lliu315}@gatech.edu jx@cc.gatech.edu
[††] Peking University, Beijing, China
{liuziheng,yang.tong}@pku.edu.cn
[‡]University of Miami, Coral Gables, USA
ogihara@cs.miami.edu

ABSTRACT

Set reconciliation is a fundamental algorithmic problem that arises in many networking, system, and database applications. In this problem, two large sets A and B of objects (bitcoins, files, records, etc.) are stored respectively at two different network-connected hosts, which we name Alice and Bob respectively. Alice and Bob communicate with each other to learn $A \triangle B$, the difference between A and B, and as a result the reconciled set $A \bigcup B$.

Current set reconciliation schemes are based on either invertible Bloom filters (IBF) or error-correction codes (ECC). The former has a low computational complexity of $O(d)$, where d is the cardinality of $A \triangle B$, but has a high communication overhead that is several times larger than the theoretical minimum. The latter has a low communication overhead close to the theoretical minimum, but has a much higher computational complexity of $O(d^2)$. In this work, we propose Parity Bitmap Sketch (PBS), an ECC-based set reconciliation scheme that gets the better of both worlds: PBS has both a low computational complexity of $O(d)$ just like IBF-based solutions and a low communication overhead of roughly twice the theoretical minimum. A separate contribution of this work is a novel rigorous analytical framework that can be used for the precise calculation of various performance metrics and for the near-optimal parameter tuning of PBS.

PVLDB Reference Format:
Long Gong, Ziheng Liu, Liang Liu, Jun Xu, Mitsunori Ogihara, and Tong Yang. Space- and Computationally-Efficient Set Reconciliation via Parity Bitmap Sketch (PBS). PVLDB, 14(4): 458 - 470, 2021.
doi:10.14778/3436905.3436906

1 INTRODUCTION

Set reconciliation is a fundamental algorithmic problem that has received considerable research attention over the past two decades [10, 11, 14, 19, 22]. In the simplest form of this problem, two large sets A and B of objects (bitcoins, files, records, etc.) are stored respectively at two different network-connected hosts, which we name Alice and Bob respectively. Alice and Bob communicate with each other to find out the difference between A and B, defined as $A \triangle B \triangleq (A \setminus B) \bigcup (B \setminus A)$, so that both Alice and Bob obtain the set union $A \bigcup B$ $(= A \bigcup (A \triangle B) = B \bigcup (A \triangle B))$.

Proceedings of the VLDB Endowment, Vol. 14, No. 4 ISSN 2150-8097.
doi:10.14778/3436905.3436906

Set reconciliation arises in many networking, system, and database applications. In cloud storage systems (*e.g.,* Dropbox, Microsoft OneDrive, Google Drive, and Apple iCloud), sets of files and directories need to be synchronized across the copies stored locally on different devices and in the cloud. In distributed database systems (*e.g.,* Spanner [9] and Cassandra [16]), an update at a single node has to get replicated across all other nodes eventually. In blockchains [24, 25], transactions need to be synchronized with some peers.

1.1 Problem Formulation

As is standard in the literature, in the rest of the paper we describe only unidirectional set reconciliation, in which Alice learns $A \triangle B$ and then infers $A \bigcup B$; for bidirectional set reconciliation, Alice can simply infer $A \setminus B$ (from $A \triangle B$) and send it to Bob, from which Bob can infer $A \bigcup B$ as well. A simple but naive set reconciliation scheme is for Bob to send B, in its entirety, to Alice. This scheme, however, is grossly inefficient when $A \triangle B$ is small (in cardinality) relative to their union $A \bigcup B$, which is indeed a usual situation in most applications. In this situation, it would be ideal if only the elements (objects) in $A \triangle B$ need to be transmitted. In other words, Bob sends only $B \setminus A$ to Alice.

In the set reconciliation problem, we usually assume each element is "indexed" by a fixed-length (hash) signature, so the universe $\mathcal{U}$ contains all binary strings of this length. Let $d \triangleq |A \triangle B|$ denote the cardinality of the set difference. It is not hard to prove (using information theory) that the theoretical minimum amount of communication between Alice and Bob for the *bidirectional* set reconciliation is the size of the set difference $d \log |\mathcal{U}|$ [22]. It is reasonable to use this minimum as a comparison benchmark for communication overheads in the unidirectional case (wherein Alice learns $A \triangle B$), because it is provably also the minimum for this unidirectional case in certain worst-case scenarios such as $A \subset B$. Hence we will do so throughout this paper.

1.2 Existing Approaches

Although many techniques have been proposed for this problem, they all fall victim to a seemingly fundamental tradeoff between the communication overhead, of transmitting the codewords (in the general sense rather than in the narrow context of error-correction codes) needed for set reconciliation, and the computational complexity of decoding such codewords. The majority of such techniques are based on either invertible Bloom filters (IBF) or error-correction codes (ECC). On one hand, IBF-based techniques incur a communication overhead that is several times (e.g., 6 times in [11]) the

theoretical minimum $d \log |\mathcal{U}|$, but have a linear (i.e., $O(d)$) decoding computational complexity. On the other hand, ECC-based techniques have a low communication overhead close to the theoretical minimum, but have a decoding computational complexity of $O(d^2)$ finite field operations, which can be very high when d is large (say $d = 10{,}000$).

1.3 Our Solution

In this work, we propose a solution, called Parity Bitmap Sketch (PBS), that mostly avoids this unfortunate tradeoff and gets the better of both worlds. More specifically, PBS has both a low computational complexity of $O(d)$ just like IBF-based solutions and a low communication overhead of roughly twice the theoretical minimum. PBS also has another advantage over all existing solutions in that it is "piecewise reconciliable" in the following sense. In all existing solutions, decoding of the codewords to obtain $A \triangle B$ is an all-or-nothing process in the sense that when the decoding failed (albeit usually with a small probability when the codewords are appropriately parameterized), little knowledge has been learned (so most of the communication, encoding, and decoding efforts are wasted) and the process starts from square one. In contrast, in PBS, the decoding of each codeword (also in the general sense) is independent of those of others, and the successful decoding of each codeword leads to a subset of *distinct elements* being reconciled; here and in the sequel, we refer to each element in $A \triangle B$ a *distinct element*. This way, additional efforts are incurred only for the small percentage of codewords whose decodings failed earlier.

Here we sketch the main ideas of PBS, by assuming that the set difference cardinality d is small (say no more than 5 elements), and the value of d is precisely known; both assumptions will be removed later in the paper. The first step of PBS is to partition A and B each into subsets in a consistent manner. We partition the set A into n disjoint subsets A_1, A_2, ..., A_n using a hash function h as follows: Each A_i, $i = 1, 2, \cdots, n$, contains all elements in A that are hashed to value i (by h). We similarly partition B into $B_1, B_2, \cdots, B_n$ using the same h. The use of a common h induces a hash-partitioning (also by h) of the set-pair-difference $A \triangle B$ into n disjoint subset-pair-differences $A_1 \triangle B_1$, $A_2 \triangle B_2$, ..., $A_n \triangle B_n$. We set this constant n to be roughly an order of magnitude larger than d^2, so that with high probability, the following *ideal situation* happens: The d distinct elements between A and B are hashed (by h) into d distinct subset-pair-differences, so that each such subset-pair-difference contains exactly one distinct element. This situation is ideal, because each such subset pair can be easily reconciled as will be shown in §2.1.

The second step of PBS is to encode partitions $\{A_i\}_{i=1}^n$ and $\{B_i\}_{i=1}^n$ each into an n-bit-long *parity bitmap*. The n-bit-long parity bitmap encoding of $\{A_i\}_{i=1}^n$, denoted as $A[1..n]$, is defined as follows. For $i = 1, 2, ..., n$, $A[i]$, the i^{th} bit of $A[1..n]$, is equal to 1 if A_i contains an odd number of elements, and is equal to 0 otherwise. The n-bit-long parity bitmap of $\{B_i\}_{i=1}^n$, denoted as $B[1..n]$, is similarly defined. In the aforementioned ideal situation of the d elements in $A \triangle B$ landing in d distinct subset-pair-differences, the two bitmaps differ in exactly d bit positions. In this situation, if Bob knows these d bit positions then the d corresponding subset pairs, and hence the set pair A and B, can be easily reconciled as we will describe in §2.2.

While Alice can certainly send the n-bit-long parity bitmap $A[1..n]$ to Bob, this is quite wasteful since $n \gg d$. A more communication-efficient way, introduced first in PinSkech [10], is to view $B[1..n]$ as a "corrupted" copy of $A[1..n]$ that contains d "bit errors" at the d bit positions where $A[1..n]$ and $B[1..n]$ differ, and to let Alice send Bob instead a BCH [5] codeword of much shorter length to "correct" these "bit errors". Referring to this BCH coding as sketching (as was done in [10]), we call our scheme Parity Bitmap Sketch (PBS).

Another significant contribution of this work is a rigorous and accurate Markov-chain modeling of the multi-round set reconciliation process of PBS. This model enables not only the accurate analysis of various performance metrics, such as the probability that all distinct elements are successfully reconciled in r rounds, but also the tuning of the parameters of PBS for near-optimal performances.

1.4 Possible Applications of PBS

As explained earlier, elements in set difference $A \triangle B$ are the hash signatures of actual objects that need to be exchanged. When the size of an object is much larger than that of a hash signature, the communication overhead of reconciling A and B, using any existing set conciliation scheme except the naive scheme, is anyway negligible compared to that of exchanging the actual objects. However, in many real-life applications, either the actual object size is not significantly larger (e.g., in the transaction relay operation of a blockchain scheme), or the actual objects need to be synchronized much less often than their hash signatures (e.g., in Dropbox under the smart sync mode [1]). In such applications, it makes a performance difference to reduce the communication overhead of reconciling A and B.

For example, as measured in a blockchain work called Erlay [24], this communication overhead accounts for around 5% of the total network bandwidth consumption of its transaction relay operation. In this case, PinSkech [10], the most communication-efficient set reconciliation scheme, is used and the size of the hash signature (called *transaction ID* in blockchain schemes) is compressed from 256 bits to 64 bits (at cost of possible hash collisions among different transactions during the set reconciliation process). This communication overhead would increase to around 55% of the total bandwidth consumption if IBF-based schemes were used instead and transaction ID's were not compressed.

PBS is better suited, than any existing set reconciliation scheme, for such applications in general and blockchain schemes in particular, for two reasons. First, although PinSketch, the state-of-the-art ECC-based solution, has a slightly smaller communication overhead than PBS, its computational complexity is too high to scale to scenarios where $|A \triangle B|$ is large. Second, the communication overhead of IBF-based solutions, including the state-of-the-art solution Graphene [25], are generally much larger than that of PBS.

Therefore, in the following we will use the transaction relay operation in blockchain schemes as an example application for PBS. Transaction relay (operation) refers to the synchronization (reconciliation) of the transaction databases (sets) across the peer-to-peer network of a blockchain scheme. In this application, Alice and Bob are two peers engaging in a transaction relay, and A and B

are the sets of hash signatures of the transactions recorded at Alice and Bob respectively.

The rest of the paper is organized as follows. First, we describe the PBS scheme for a small d value and that for a large d value in §2 and §3, respectively. Then, we describe our analytical framework in §4 and apply it to the performance analysis and the near-optimal parameter tuning of PBS in §5. After that, we present a new estimator for estimating the set difference cardinality in §6. Finally, we survey existing set reconciliation schemes in §7, compare the performance of PBS with that of some of them in §8, and conclude the paper in §9.

2 PBS FOR SMALL d

In this section, we describe how the PBS scheme allows Alice and Bob to reconcile their respective sets A and B, where $d = |A \triangle B|$ is assumed to be small and precisely known. We start with the trivial case where $d \leq 1$ in §2.1 and then generalize the scheme for the case where d is a small number in §2.2. As will be explained later in §2.2.2, the latter will use the former as a building block.

2.1 The Trivial Case of $d \leqslant 1$

1 *Bob*: $s_B \leftarrow \bigoplus_{b \in B} b$; Send s_B to Alice;
2 *Alice*: $s_A \leftarrow \bigoplus_{a \in A} a$; $s \leftarrow s_A \oplus s_B$;

Procedure 1: Set reconciliation when $d \leq 1$

Procedure 1 shows the set reconciliation scheme for the trivial case, in which A and B differ by at most one (distinct) element. It consists of two steps. First, Bob calculates the XOR sum s_B, the bitwise-XOR of all elements in B, and sends it to Alice. Second, Alice calculates s_A, the XOR sum of all elements in A. Upon receiving s_B from Bob, Alice computes $s \triangleq s_A \oplus s_B$. The value of s tells Alice which of the following two cases happens.

- Case (I): If $s = \mathbf{0}$, which implies $s_A = s_B$, Alice concludes that A and B have no distinct element or $A = B$. Here $\mathbf{0}$ denotes the $\log|\mathcal{U}|$-bit-long all-0 string;
- Case (II): If $s \neq \mathbf{0}$, which implies $s_A \neq s_B$, Alice concludes that A and B have exactly one distinct element which is s, i.e., $A \triangle B = \{s\}$. This is because XORing s_A and s_B (to obtain s) cancels out all (common) elements in $A \bigcap B$.

Like in most of the literature on set reconciliation, we assume that the all-0 element $\mathbf{0}$ is excluded from the universe $\mathcal{U}$, since otherwise Procedure 1 does not work for the following reason. When the computed s is $\mathbf{0}$, Alice cannot tell whether A and B are identical, or they have $\mathbf{0}$ as their distinct element. Procedure 1 also does not work when there are more than one distinct elements in $A \triangle B$ (i.e., $d > 1$), since the computed s in this case is the XOR sum of all these distinct elements.

2.2 The General Case

In this section, we describe the scheme for the more general case where d is a small number (say 5), but is not necessarily 0 or 1.

2.2.1 Hash-partitioning and parity bitmap encoding. Here we formalize the aforementioned process of partitioning A into $\{A_i\}_{i=1}^n$, B into $\{B_i\}_{i=1}^n$, and $A \triangle B$ into $\{A_i \triangle B_i\}_{i=1}^n$ using a hash function h. Define *sub-universe* $\mathcal{U}_i$ as the set of elements in the universe $\mathcal{U}$ that are hashed into value i. More precisely, $\mathcal{U}_i \triangleq \{s \,|\, s \in \mathcal{U} \text{ and } h(s) = i\}$ for $i = 1, 2, ..., n$. Then defining $A_i \triangleq A \bigcap \mathcal{U}_i$ and $B_i \triangleq B \bigcap \mathcal{U}_i$ for $i = 1, 2, ..., n$ induces the partitioning of A, B, and $A \triangle B$.

How the d distinct elements (balls) in $A \triangle B$ are "scattered" into the n subset-pair-differences (bins) $\{A_i \triangle B_i\}_{i=1}^n$ can be precisely modeled as throwing d balls each uniformly and randomly into one of the n bins. For the moment, we assume the following *ideal case* happens: Every subset-pair-difference $A_i \triangle B_i$ contains at most one distinct element. This ideal case corresponds to the d balls ending up in d distinct bins. It happens with probability $\prod_{k=1}^{d-1}(1 - \frac{k}{n})$, which is on the order of $1 - O(d^2/n)$ when $n \gg d$. Hence, n must be $\Omega(d^2)$ to ensure the ideal case happens with a nontrivial probability, as mentioned earlier in §1.3.

1 *Alice*: Send ξ_A, the BCH codeword of $A[1..n]$, to Bob;
2 *Bob*: Decode $B[1..n] \| \xi_A$ to obtain $i_1, i_2, ..., i_d$;
3 *Bob*: Send XOR sums of sets $B_{i_1}, B_{i_2}, ..., B_{i_d}$ (Line 1 in Procedure 1), indices $i_1, i_2, ..., i_d$, and checksum $c(B)$ to Alice;
4 *Alice*: Obtain $A_{i_1} \triangle B_{i_1}, A_{i_2} \triangle B_{i_2}, ...,$ and $A_{i_d} \triangle B_{i_d}$ (Line 2 in Procedure 1); $\hat{D} \leftarrow \bigcup_{k=1}^{d} (A_{i_k} \triangle B_{i_k})$;
5 *Alice*: Check whether $c(A \triangle \hat{D}) \stackrel{?}{=} c(B)$;

Procedure 2: PBS-for-small-d (first round)

2.2.2 Find and reconcile the d subset pairs. The remaining steps of the PBS scheme are summarized in Procedure 2. Recall that the partitions $\{A_i\}_{i=1}^n$ and $\{B_i\}_{i=1}^n$ can be encoded as parity bitmaps $A[1..n]$ and $B[1..n]$ respectively, in which each $A[i]$ or $B[i]$ corresponds to the parity (oddness or evenness) of the cardinality of the subset A_i or B_i. In the ideal case, $A[1..n]$ and $B[1..n]$ differ in exactly d distinct bit positions. Suppose these d bit positions are $i_1, i_2, ..., i_d$. Then subset pairs $(A_{i_1}, B_{i_1}), (A_{i_2}, B_{i_2}), ..., (A_{i_d}, B_{i_d})$ each differs by exactly 1 (distinct) element. Hence each subset pair can be reconciled using Procedure 1.

For this to happen, however, both Alice and Bob need to first know the values of $i_1, i_2, ..., i_d$, or the bit positions where $A[1..n]$ and $B[1..n]$ differ. To this end, a naive solution is for Alice to send $A[1..n]$ to Bob and for Bob to compare it with $B[1..n]$. However, as mentioned earlier in §1.3, Alice can achieve the same goal by sending an ECC codeword ξ_A that is much shorter than $A[1..n]$. The idea is that since $B[1..n]$ (which Bob already knows) can be viewed as a "corrupted" (with d bit errors in the positions $i_1, i_2, ..., i_d$) copy of $A[1..n]$, as long as the codeword ξ_A is parameterized to correct at least d bit errors, Bob can decode $B[1..n] \| \xi_A$ (the "corrupted" message concatenated with the ECC codeword of the "uncorrupted" message) to find out the positions of these d bit errors.

Although several ECC schemes are suitable for this purpose, we choose the BCH scheme for PBS because it results in near-optimal codeword length in the following sense: In the context of PBS, to "correct up to t bit errors", ξ_A only needs to be $t\lceil\log(n+1)\rceil$ bits

long; even if Alice knew these t bit positions precisely, specifying each bit position (to Bob) would require $\lceil \log n \rceil$ bits. BCH is also the choice of PinSkech [10] for the same reason.

Once Bob decodes $B[1..n]||\xi_A$ (Line 2 in Procedure 2) to obtain $i_1, i_2, ..., i_d$, Bob sends the XOR sums of the corresponding subsets B_{i_1}, B_{i_2},...,B_{i_d} to Alice (Line 1 in Procedure 1). Bob also needs to send the decoded "bit error positions" $i_1, i_2, ..., i_d$ to Alice (Line 3 in Procedure 2), since Alice cannot obtain this information by herself without knowing anything about $B[1..n]$. In addition, for Alice to verify whether the set reconciliation is successfully completed (to be described next), Bob sends $c(B)$, a checksum of its set B, to Alice.

2.2.3 Verify the estimated set difference. Once Alice receives the "bit error positions" and the corresponding XOR sums, she can recover the distinct elements each using Procedure 1 to arrive at the estimated set difference (Line 4 in Procedure 2), which we denote as $\hat{D}$. It is not hard to verify that in the ideal case this $\hat{D}$ is necessarily the same as the actual set difference $A \triangle B$, so the unidirectional set reconciliation process is successfully completed.

However, the nonideal case can happen and when that happens $\hat{D}$ is in general not the same as $A \triangle B$. Hence, Alice in general needs to verify whether $\hat{D} \stackrel{?}{=} A \triangle B$ after a round of set reconciliation process. Alice does so by checking an equivalent condition $A \triangle \hat{D} \stackrel{?}{=} B$ as follows. She applies a checksum function $c(\cdot)$ to $A \triangle \hat{D}$ and comparing (Line 5 in Procedure 2) the resulting checksum $c(A \triangle \hat{D})$ with $c(B)$ that Alice received earlier from Bob. We use as $c(\cdot)$ here the plain-vanilla summation function, with which the checksum of a set S is the sum of all elements (viewed as integers) modulo $|\mathcal{U}|$. The length of such a checksum is $(\log |\mathcal{U}|)$ bits, the same as that of an element. We use this checksum function for two reasons. First, because it uses the '+' operation whereas the set reconciliation process (Procedure 1) involves a very different operation (XOR), a false verification event is intuitively almost statistically uncorrelated[1] with any reconciliation error (called an exception and to be described shortly) event, which makes the verification step meaningful and effective to the maximum extent. Second, this checksum function can be incrementally computed.

Using a 32-bit-long checksum (assuming $\log |\mathcal{U}| = 32$), the probability for Alice to mistakenly believe $A \triangle \hat{D} = B$ when the opposite (i.e., $\{A \triangle \hat{D} \neq B\}$) is true is only $O(10^{-12})$ for the following reason. The false verification event $\{A \triangle \hat{D} \neq B\}$ can happen only in the nonideal case, which happens with a probability of $O(10^{-2})$ (as we will show in §2.3). Then, conditioned upon the event $\{A \triangle \hat{D} \neq B\}$ happening, the probability for their 32-bit-long checksums happen to be equal (i.e., $c(A \triangle \hat{D}) = c(B)$) is only $2^{-32} \approx 2.3 \times 10^{-10}$. This $O(10^{-12})$ probability of incorrect verification should be acceptable in most applications.

In applications in which correct verification absolutely has to be guaranteed (*e.g.,* bitcoin), additional built-in verification mechanisms, such as Merkle tree, are usually used, which can reduce the probability of false verification to practically zero at no extra cost (to PBS). For example, blockchain platforms Ethereum [2] and Bitcoin [23] both have Merkle tree [20] based mechanisms for verifing the integrity and the consistency of transactions. A Merkle tree is a binary tree in which a parent node digitally certifies (verifies) its two children. In the cases of Ethereum and Bitcoin, each transaction corresponds to a leaf node of the Merkle tree that records the cryptographic hash value of the transaction, and each non-leaf node records the cryptographic hash value of its two children. This way, the root node digitally certifies the integrity and the consistency of all transactions. For mission-critical applications that do not have such an additional built-in verification mechanism, we can add one at a small cost. For example, we can compute and check $H(A \triangle \hat{D}) \stackrel{?}{=} H(B)$, where H is a one-way multiset hash function such as MSet-XOR-Hash [7], at the additional cost of $O(\max\{|A|+d, |B|\})$ computation overhead and constant communication overhead.

In the case of PBS-for-small-d, the set reconciliation process will run as many rounds as it takes (to be explained in §2.4) for the checksums of two sets being reconciled to eventually match each other; in the case of PBS-for-large-d, the same can be said about the set reconciliation process for each group pair (to be explained in §3). Hence, barring the false verification event, which as just explained happens with $O(10^{-12})$ probability when using only a 32-bit checksum or with practically zero probability when using additional cryptographic verification techniques, the set reconciliation process (for both large and small d) guarantees to correctly reconcile A and B (and the respective referenced objects) when it halts. The formal proof can be found in Appendix D in [12]. With this understanding, for ease of presentation, we assume in the sequel that a false verification will never happen in the checksum verification step.

In Sections 2.3 and 3.2, we describe three types of exceptions may result in a $\hat{D}$ that is incorrect (not the same as $A \triangle B$). When that happens, the checksum verification step will not accept $\hat{D}$ as is, as just explained. Hence, these exceptions will never result in an incorrect set reconciliation. They can only delay the inevitable (eventual correct reconciliation of A and B) by triggering additional rounds of set reconciliation process. We note there is no need for PBS to determine which bin or bins cause the checksum verification step to fail in the current round, because as we will show in §2.4 such information is not used anywhere in the next round of set reconciliation operation.

2.3 Exception Handling

When the ideal case does not happen, some subset pairs may contain more than one distinct elements and cannot be successfully reconciled by Procedure 1. In this case, the checksum verification step will detect this event and trigger another round of PBS to reconcile the "remaining" distinct elements, as will be elaborated in §2.4. Two types of exceptions can possibly happen in such a subset pair, say (A_i, B_i).

Type (I) exception: $A_i \triangle B_i$ contains a nonzero even number of distinct elements. In this case $A[i] = B[i]$ since the cardinalities of A_i and B_i are either both even or both odd. The BCH codeword ξ_A cannot detect this exception. This exception happens with a small but nontrivial probability. For example, when $d = 5$ and $n = 255$ (i.e., throwing 5 balls each uniformly and randomly into 255 bins), the probability that some bin has a nonzero even number of (in this case either 2 or 4) balls is roughly 0.04.

[1] If we absolutely have to ensure any two such events to be provably strictly statistically uncorrelated, we can apply a one-way hash function to each element first and adding their hash values (viewed as integers) up instead, at a bit extra computation cost.

Type (II) exception: $A_i \triangle B_i$ contains an odd number (at least 3) of distinct elements. In this case, $A[i] \neq B[i]$. Bob will mistakenly believe that (A_i, B_i) contains exactly one distinct element and try to recover it using Procedure 1. The "recovered" element s is however the XOR sum of all distinct elements in $A_i \triangle B_i$ as explained at the end of §2.1. We call this s a fake distinct element. This event happens with a tiny probability. In the same example above ($d = 5$ and $n = 255$), the probability that some bin has an odd number of balls (in this case either 3 or 5) is only 1.52×10^{-4}. This probability can be further reduced, thanks to the consistent nature of hash-partitioning, which provides us with a no-cost mechanism that can detect fake distinct elements (so that they will not be included in $\hat{D}$) with high probability. We describe this detection mechanism in Appendix C in [12].

2.4 Running PBS for Multiple Rounds

As mentioned earlier, when the ideal case does not happen, Alice and Bob cannot successfully reconcile their respective sets A and B in a single round, and Alice can tell this situation from the checksum verification step. In this situation, Alice and Bob need to run additional rounds of Procedure 2, but with a different input set pair (than (A, B)) as follows. Let $\hat{D}_1$ be the estimated set difference Alice obtained in the first round. In the second round, Alice and Bob try to reconcile their respective sets $A \triangle \hat{D}_1$ and B, from which Alice obtains another estimated difference (between $A \triangle \hat{D}_1$ and B) that we denote as $\hat{D}_2$. If the set reconciliation is still not successfully completed, Alice and Bob run a third round to try to reconcile sets $(A \triangle \hat{D}_1) \triangle \hat{D}_2$ and B. This process continues until the set reconciliation is successfully completed as verified by the checksum. The final output of the process, which is what Alice believes to be $A \triangle B$, is $\hat{D}_1 \triangle \hat{D}_2 \triangle \cdots \triangle \hat{D}_r$, where r is the number of rounds this process runs and $\hat{D}_i$ for $i = 1, 2, \cdots, r$ is the estimated set difference in the i^{th} round.

In each subsequent round, a different and (mutually) independent hash function is used to perform the consistent hash partitioning of the two sets to be reconciled (e.g., $A \triangle \hat{D}_1$ and B in the second round), so that the same type (I) and/or (II) exceptions encountered in the previous round, which have so far prevented the set reconciliation from being successfully completed, can be avoided with overwhelming probability. The use of independent hash functions in different rounds offers another significant benefit: How the number of unreconciled distinct elements decreases one round after another (and eventually goes down to 0) can now be precisely modeled as a Markov chain, as will be elaborated in §4.

2.5 BCH Encoding and Decoding

In this section, we describe the specific BCH encoding and decoding in PBS; how this encoding differs from that for its usual application of communication over a noisy channel can be found in Appendix H in [12]. Recall that in Line 1 of Procedure 2, Alice sends, instead of the "message" $A[1..n]$ itself, its much shorter BCH codeword ξ_A to Bob. We define the *error-correction capacity* of an ECC codeword as the maximum number of bit errors it can correct. In the case of PBS-for-small-d, where d is assumed to be known precisely beforehand, the error-correction capacity of ξ_A is set to d so the BCH decoding is always successful. However, as will be explained in §3.1, when d is large and the sets A and B each has to be partitioned into groups, the number of "bit errors" that occur to a group pair can exceed the error-correction capacity of the corresponding BCH codeword. In this case, a BCH decoding failure will happen and how to deal with its fallout will be explained in §3.2.

We now briefly explain what is involved for Bob to decode the BCH codeword ξ_A against its local bitmap $B[1..n]$. Here the only task is to figure out the "bit error positions" (in which $A[1..n]$ and $B[1..n]$ differ). To do so, Bob needs to invert a $d \times d$ matrix in which each matrix entry is an element of the finite field $GF(2^m)$ where $m = \lceil \log(n+1) \rceil$. In PBS, n is always set to $2^m - 1$ for some positive integer m in BCH codes for achieving the maximum coding efficiency. Hence, we drop "floor" and "ceiling" and consider $m = \log n$ in the sequel. Normally such a matrix inversion would take $O(t^3)$ finite field operations over $GF(2^m)$. However, since this matrix takes a special form called Toeplitz, it can be inverted in $O(d^2)$ operations over $GF(2^m)$ using the Levinson algorithm [17].

3 PBS FOR LARGE d

In this section, we continue to assume that the number of distinct elements d is precisely known in advance. The PBS-for-small-d scheme just described is no longer suitable when d is very large, since its BCH decoding computational complexity is $O(d^2)$ finite field operations. Instead, in this case we partition, consistently using a different hash function h' (than the h above), sets A and B each into d/δ smaller sets, where δ is a small number (just like what we earlier assumed d to be). We refer to these smaller sets as *groups* to distinguish them from the *subsets* A_i's and B_i's above. We then apply PBS-for-small-d to each of the g group pairs.

Here δ is the average number of distinct elements per group pair. It is a tunable parameter, by which we can control the tradeoff between the communication and the computational overheads of PBS. In general, the larger δ is, the lower the communication overhead and the higher the computational overhead are. We have elaborated in Appendix I.1 in [12] how δ controls this tradeoff. Since δ=5 appears to be a nice tradeoff point, we fix the value of δ at 5 in this paper. Since each group pair contains on average $\delta = 5$ distinct elements, the BCH decoding computational complexity per group pair can be considered $O(1)$. As a result, the overall BCH decoding computational complexity is $O(d)$ for all $g = d/\delta$ group pairs. We refer to this PBS-for-large-d scheme as PBS in the sequel except in places where this abbreviation would result in ambiguity or confusion.

3.1 How to Set Parameters t and n

In PBS (i.e., PBS-for-large-d), we have to make some design decisions that we don't have to in PBS-for-small-d. One of them is how to set the error-correction capacities of the BCH codes used for each of the g group pairs. Let δ_i, $i = 1, 2, ..., g$, be the number of distinct elements that group pair i have. If we knew the precise values of $\delta_1, \delta_2, ..., \delta_g$, we would simply set the BCH error-correction capacity for each group pair i, which we denote as t_i, to δ_i. This way, each BCH codeword is the shortest possible for the respective task, which minimizes the communication overhead of transmitting these codewords. In reality, we do not know the exact value of any δ_i, since it is a random variable with distribution $Binomial(d, 1/g)$

thanks to the hash-partitioning (of A and B each into g groups); we only know that $E[\delta_i] = d/g = \delta$ but that does not help much. In theory, we can measure δ_i using a (set difference) cardinality estimation protocol. However, as will be shown in §6, to obtain such an estimate using the best protocol would incur hundreds of bytes of communication overhead. In comparison, the "savings" on the communication overhead that such an estimate would bring (for the corresponding group pair) is only tens of bytes, as we will elaborate next.

In PBS, we set an identical BCH error-correction capability t for all g group pairs. It intuitively makes sense since random variables δ_1, δ_2, ..., and δ_g are identically distributed. Now the next question is "How should we set this t?". This is a tricky question because, on one hand, if t is too large (say several times larger than δ), then the total size of the BCH codewords is unnecessarily large, resulting in "wastes" in communication overhead; but on the other hand, if t is too small (say equal to δ), then a large proportion of the BCH codewords cannot decode, resulting in considerable additional efforts and costs (i.e., "penalties") for reconciling the large proportion of affected group pairs. In §4, we propose an analytical framework that can be used to identify the t value that minimizes "wastes + penalties" (in §5.1). This optimal t value can range from 1.5δ to 3.5δ depending on how large this d is.

Based on a similar rationale, we set another parameter for each group pair i to the same value n: Each group pair i is to be partitioned into n subsets, so that the parity bitmaps ($A[1..n]$ and $B[1..n]$ in PBS-for-small-d) for all groups have the same length of n bits. This n is also a tunable parameter (for optimal PBS performance), since the probability for the ideal case (of all distinct elements between a group pair being hashed to distinct subsets) to happen is a function of n and $\delta = 5$. As will be elaborated in §5.1, our analytical framework can also be used for the optimal tuning of n.

Communication Overhead Per Group Pair. Here we analyze the total communication overhead of the first round of PBS. Since the vast majority of distinct elements are discovered and reconciled in the first round, as will be shown in §5.3, it represents the vast majority (over 95%) of that over all rounds. For each group pair i, the communication overhead (of running PBS-for-small-d on this pair) in the first round contains the following four components: (1) the BCH codeword that is $t \log n$ bits long; (2) the δ_i "bit error locations" whose total length is $\delta_i \log n$ bits; (3) the δ_i `XOR` sums whose total length is $\delta_i \log |\mathcal{U}|$ bits; and (4) the checksum that is $\log |\mathcal{U}|$ bits long. Hence the average communication overhead of PBS per group pair in the first round is

$$t \log n + \delta \log n + \delta \log |\mathcal{U}| + \log |\mathcal{U}| \quad (1)$$

3.2 Exception Handling on BCH Decoding

Recall that in PBS-for-small-d we need to handle two types of exceptions: type (I) and type (II). In PBS-for-large-d, we have another exception to worry about. This exception arises when the number of bit positions where bitmaps A and B in Procedure 2 differ is larger than t, the universal BCH error-correction capability (for every group pair). When this exception happens, the BCH decoding would fail (when executing Line 2 in Procedure 2) and the decoder would report a failure. With t appropriately parameterized as explained earlier, this exception should happen with a small probability to any group pair. For example, when d = 1,000, $\delta = 5$ (so that $g = d/\delta = 200$), and t is set to the optimal value of 13 ($= 2.6\delta$), the probability for this exception to happen to any group pair is only 6.7×10^{-4}.

To handle this exception, we further hash-partition each trouble-causing group pair (whose BCH decoding has failed) into 3 sub-group-pairs and reconcile each of them using PBS-for-small-d. With this three-way split, with an overwhelming probability, each sub-group-pair should contain no more than t distinct elements and its BCH decoding operation should be successful in the next round. For example, when $\delta = 5$ and $t = 13$ (same as in the example above), the conditional (upon the occurrence of this exception) probability for any sub-group-pair to contain more than t distinct elements is only 9.5×10^{-10}. We use a three-way split here because a two-way split would result in a much higher conditional probability (0.0012 in this example) for this event. All said, if necessary, a trouble-causing sub-group-pair will be further split three-way.

As explained earlier, the ultimate gatekeeper for ensuring the correctness of set reconciliation is the checksum verification step, which in this case (of large d) is applied to each group pair. BCH decoding exceptions alone, or in combination with type (I) or (II) exceptions, may only delay the inevitable eventual correct reconciliation of A and B, as long as a false checksum verification event does not happen.

3.3 Multi-round Operations

In PBS, the set reconciliation processes of the g group pairs are independent of each other. Each group pair runs as many rounds of PBS-for-small-d as needed to reconcile all distinct elements between them. Almost every set reconciliation scheme is designed and parameterized to provide the performance guarantee that the reconciliation process is successfully completed, in the sense all distinct elements are correctly reconciled, with at least a target probability p_0. In PBS, this guarantee will involve an additional parameter r that is the target number of rounds the scheme is allowed to run to reach this target success probability. More precisely, the multi-group-pair multi-round operation of PBS must, with a probability that is at least p_0, be successfully completed in r rounds.

Let R be the number of rounds it takes for all g group pairs, and hence the set pair, to be successfully reconciled. This guarantee can then be succinctly written as $Pr[R \leq r] \geq p_0$. Intuitively, we can always provide this guarantee by making the values of the two key parameters n (the size of the parity bitmap) and t (the BCH error-correction capacity) very large, but doing so would result in a high communication overhead. This apparent tradeoff leads us to study the following parameter optimization problem: Among all parameter settings of n and t that can guarantee $Pr[R \leq r] \geq p_0$, which one results in the smallest communication overhead?

To tackle this optimization problem, we need to first analyze the multi-group-pair success probability $Pr[R \leq r]$. The latter boils down roughly (but not exactly as we have explained in Appendix G in [12]) to analyzing the following single-group-pair success probability. Consider a single group pair that have x distinct elements between them before the first round starts. For the moment, we assume $x \leq t$ so that we do not have to worry about the BCH

decoding failure. Our problem is to derive the formula for the probability of the following event that we denote as $\{x \overset{r}{\rightsquigarrow} 0\}$: All the x distinct elements, and hence the pair, are successfully reconciled in no more than r rounds. Solving this problem is the sole topic of §4.

4 ANALYTICAL FRAMEWORK

In this section, we derive a Markov-chain model for analyzing the aforementioned single-group-pair success probability $Pr[x \overset{r}{\rightsquigarrow} 0]$. We will show next that, under this model, the initial state of the Markov chain is state x (distinct elements), and each (set reconciliation) round triggers a state transition. Hence, the event $\{x \overset{r}{\rightsquigarrow} 0\}$ corresponds to the Markov chain reaching the "good" state 0 (distinct elements left) within r transitions. Suppose the transition probability matrix of this Markov chain is M. The formula for computing the probability of this event is simply

$$Pr[x \overset{r}{\rightsquigarrow} 0] = \left(M^r\right)(x, 0) \tag{2}$$

wherein $\left(M^r\right)(x, 0)$ is the element at the intersection of the x^{th} row and the 0^{th} column in the matrix M^r (M to the power r).

How the number, starting at x before the first round, of yet unreconciled distinct elements between the group pair decreases one round after another, and eventually goes down to 0, can be precisely modeled as a Markov chain as follows. As described in §2.4, in the first round, each of the x balls (distinct elements) is thrown uniformly and randomly (by the hash function h) into one of the n bins (subset pairs). If a ball ends up in a bin that contains no other balls, the corresponding distinct element can be successfully reconciled using Procedure 1. We call this ball a "good" ball, since it does not have to be thrown again in later rounds, and for the modeling purpose call this bin a "good" bin (just for this round). If a ball ends up in a bin that has other balls, which corresponds to a type (I) or type (II) exception discussed earlier in §2.3, the corresponding distinct element cannot be reconciled in this round. We call this ball a "bad" ball, since it has to be thrown again in the second round, and for the modeling purpose call this bin a "bad" bin (again just for this round).

As described in §2.4, the "bad" balls (if any) that remain after the first round will be thrown again in the second round, the "bad" balls (if any) that remain in the second round will be thrown again in the third round, and so on. Let D_k, $k = 1, 2, \cdots$, be the number of balls that remain "bad" (yet unreconciled distinct elements) after the k^{th} round. Let $D_0 = x$ be the number of balls to be thrown at the beginning (i.e., right before the first round). Then $\{D_k\}_{k=0}^{\infty}$ is a Markov chain for the following reason. Since a different and mutually independent hash function is used in each round, the random variable D_k, which is the number of balls that remain "bad" after the k^{th} round, depends only on D_{k-1}, the number of balls thrown in the k^{th} round, and is conditionally (upon D_{k-1}) independent of the history $D_0, D_1, D_2, \cdots, D_{k-2}$.

For all practical purposes (e.g., for computing $Pr[x \overset{r}{\rightsquigarrow} 0]$), M can be considered a $(t+1) \times (t+1)$ matrix, where t is the BCH error-correction capacity. To compute $M(i, j)$ is not straightforward for the following reason. Each state j with $j > 3$ in the Markov chain is a *composite state* consisting of a large number of *atom states*. Only the transition probability from state i to any atom state (of state j) can be stated as a closed-form expression (more precisely, a multinomial formula) and computed straightforwardly. The value of $M(i, j)$ is the total of all the transition probabilities from state i to each of the atom states of state j. Since the number of atom states grows exponentially with j, it is complicated (as it is necessary to enumerate all atom states), error-prone, and computationally expensive to compute $M(i, j)$ this way when j is large (say $j > 12$), as we will elaborate next.

Each atom state of state j is, in combinatorics terms, *a permutation of a combination of j*, which here corresponds to how these j balls are distributed in the n bins (by the hash function). For a simple example, when $j = 4$ and $n = 7$, the vector $(2, 0, 0, 0, 0, 2, 0)$ is such an atom state, which corresponds to these 7 bins (in a predefined order such as the natural order) having $2, 0, 0, 0, 0, 2$, and 0 "bad" balls in them respectively. Clearly, the number of such atom states (vectors) grows exponentially with j. For instance, when $j = 13, 14, 15, 16, 17$, the number of distinct atom state vectors is $2.47{\times}10^{12}, 2.10{\times}10^{13}, 1.11{\times}10^{14}, 8.03{\times}10^{14}, 4.34{\times}10^{15}$ respectively.

Our solution is to decompose each composite state j into a much small number of coarse-grained sub-states, each of which may still contain a large number of atom states. Although the transition probability from a state i to any sub-state of j is still a summation formula and hence hard to compute in the "mundane" way as explained above, we discover a recurrence relation among these transition probabilities that makes them easily computable using dynamic programming. In the interest of space, we leave out here our discussions on the preciseness of this Markov-chain model and on the detail of the dynamic programming procedure. They can be found in Appendices E and F in [12].

5 APPLYING THE FRAMEWORK

Knowing the Markov-chain model and how to compute its transition matrix M, we are now ready to tackle the aforementioned parameter optimization problem in §5.1 and study two other related parameterization and design questions in §5.2 and §5.3 respectively.

5.1 Parameter Optimization

Recall that our optimization problem is to find the optimal parameter settings of n and t that guarantee $Pr[R \le r] \ge p_0$ yet result in the smallest communication overhead. Recall that our original goal is to analyze the overall (for all g group pairs) success probability $Pr[R \le r]$. In Appendix G in [12], we have shown that $Pr[R \le r]$ is hard to calculate exactly, but can be tightly lower-bounded by $1-2(1-\alpha^g)$, where $\alpha \triangleq \sum_{x=0}^{t} Pr[X = x] \cdot Pr[x \overset{r}{\rightsquigarrow} 0]$ is a slightly underestimated success probability for any group pair, g is the number of group pairs, and t is the error-correction capacity. Here the random variable X is distributed as $Binomial(d, 1/g)$.

Minimize Communication Overhead. Armed with this rigorous lower bound on the overall success probability $Pr[R \le r]$, we can now formulate our optimization problem of parameterizing PBS to guarantee $Pr[R \le r] \ge p_0$ while minimizing the average communication overhead as follows.

$$\begin{aligned} \text{minimize} \quad & t \log n + \delta \log n \\ \text{subject to} \quad & 1 - 2(1 - \alpha^g(n, t)) \ge p_0, \quad n, t \in \mathbb{N}^+ \end{aligned}$$

The objective function $t \log n + \delta \log n$ (as a function of n and t) here is the non-constant part of the average communication overhead per group pair in the first round as shown in Formula (1). It is an appropriate objective function because it is exactly $1/g$ of the average communication overhead for all g group pairs in the first round, and as explained earlier and will be confirmed later, the first round incurs over 95% of the total communication overhead. In the constraint, we replace $Pr[R \leq r]$ by its lower bound $1-2(1-\alpha^g)$ and write α as $\alpha(n,t)$ to emphasize it is a function of n and t, when r is considered a constant. In this optimization problem, g (in the constraint) is a constant, since Alice and Bob both know d (by our assumption thus far), and $g = d/\delta$. Here δ is the average number of distinct elements per group, which we set to 5 in PBS. Hence only two variables are involved in this optimization problem: n and t.

This optimization problem is not as daunting as it might appear, since there are only a few meaningful value combinations of n and t for two reasons. First, as mentioned earlier in §2.5, n is always set to 2^m-1 for some integer m in PBS. In addition, n cannot be too small, since otherwise the ideal case (of x "balls" landing in x distinct "bins") cannot happen with high probability. The possible n values are hence narrowed down to $\{63, 127, 255, 511, 1023, 2047\}$ in practice. Second, the BCH error-correction capacity t needs to be set to between 1.5δ and 3.5δ, as explained in §3.

Our optimization procedure is simply to compute, for each of the 100 or so value combinations of n and t, the corresponding values of the lower bound $1-2(1-\alpha^g(n,t))$ (of $Pr[R \leq r]$) and the objective function $t \log n + \delta \log n$. Then among all such value combinations that can guarantee $Pr[R \leq r] \geq p_0$, we pick the one that results in the smallest objective function value.

Here we use an example to illustrate this procedure. Suppose we have $d = 1{,}000$ distinct elements, $\delta = 5$ (so that $g = 200$ groups), $r = 3$ rounds, and target success probability $p_0 = 99\%$. For each (n,t) value combination in $\{63, 127, 255, 511, 1023, 2047\} \times \{8, 9, \cdots, 16, 17\}$ we compute the corresponding lower bound $(1-2(1-\alpha^g(n,t)))$ value. The lower bound values corresponding to these (n,t) value combinations are shown Table 1. In Table 1, each cell in which the corresponding lower bound value is no smaller than the target success probability $p_0 = 99\%$ is highlighted. Among the highlighted cells, the cell that is further darkened results in the smallest objective function value and hence its "coordinates" $n = 127, t = 13$ are the optimal parameter setting in this instance. Since the matrix M can be pre-computed, the success probability value in each cell can be computed in $O(1)$ time. Hence this optimization procedure is very efficient computationally.

5.2 What If The Target r Changes?

Intuitively, when the target number of rounds r becomes smaller, it becomes more costly, in terms of both the communication and the computational (for BCH decoding) overheads, to provide the success probability guarantee $Pr[R \leq r] \geq p_0$. Intuitively, this is because n and t have to be larger so that in each group pair the ideal case happens and the BCH decoding succeeds with higher probabilities respectively. In this section, we perform a quantitative study of this tradeoff, using the same example above with $p_0 = 99\%, d = 1{,}000$ as that used in §5.1. For each $r \in \{1, 2, 3, 4\}$, we compute the optimal (n,t) value combination using the optimization procedure described above, and the corresponding optimal (minimum) average communication overhead *per group pair*.

Table 1: Success probability lower bound values.

t \ n	63	127	255	511	1023	2047
8	0	25.5%	32.7%	34.3%	34.9%	35.0%
9	52.1%	78.0%	84.2%	85.7%	86.1%	86.2%
10	75.1%	92.7%	96.5%	97.4%	97.6%	97.7%
11	85.9%	96.9%	99.1%	99.5%	99.6%	99.6%
12	91.3%	98.5%	99.7%	99.9%	>99.9%	>99.9%
13	93.9%	99.1%	99.8%	>99.9%	>99.9%	>99.9%
14	95.1%	99.4%	>99.9%	>99.9%	>99.9%	>99.9%
15	95.6%	99.5%	>99.9%	>99.9%	>99.9%	>99.9%
16	95.7%	99.6%	>99.9%	>99.9%	>99.9%	>99.9%
17	95.8%	99.6%	>99.9%	>99.9%	>99.9%	>99.9%

The optimal communication overheads per group pair are 591, 402, 318 and 288 bits when r =1, 2, 3 and 4 respectively, which confirms our earlier intuition that the larger the r is, the smaller the optimal communication overhead is. It also shows that $r = 3$ is a sweet spot: The communication overhead per group pair drops sharply from when $r = 1$ (591 bits) to when $r = 2$ (402 bits) and from when $r = 2$ to when $r = 3$ (318 bits), but drops only slightly from when $r = 3$ to when $r = 4$ (288 bits). We have found that $r = 3$ is in general a sweet spot whenever the target success probability p_0 is relatively high, such as $p_0 = 99\%$ and $p_0 = 99.58\%$ (239/240) that will be used in our evaluation. Hence we set r to 3 in this paper. For smaller p_0 values, however, $r = 2$ or even $r = 1$ can become a sweet spot, as long as d is no more than tens of millions.

5.3 Analysis on "Piecewise Reconciliability"

In this section, we perform an analysis of what portion of the distinct elements are expected to be reconciled by PBS in the first round, in the second round, and so on. Again we focus our attention on the first group pair that have δ_1 (distributed as $Binomial(d, 1/g)$ as explained earlier) distinct elements between them. Let Z_k, $k = 1, 2, ...$, be the number of distinct elements among those δ_1 that are reconciled in the k^{th} round. Our goal is to compute $\mathbf{E}[Z_1]$, $\mathbf{E}[Z_2]$, $\mathbf{E}[Z_3]$, ..., and so on. To do so, it suffices to compute the unconditional expectations $\mathbf{E}[Z_1 + Z_2 + \cdots + Z_k]$ for $k = 1, 2, \cdots$. They in turn can be derived from the following conditional expectations on the LHS of Equation (3). Equation (3) holds because both sides calculate the expected number of distinct elements that are reconciled within k rounds, conditioned upon the event $\{\delta_1 = x\}$.

$$\mathbf{E}[Z_1 + Z_2 + \cdots + Z_k | \delta_1 = x] = \sum_{y=0}^{x} (x-y) \cdot Pr[x \overset{k}{\nrightarrow} y] \qquad (3)$$

Using the above analysis, we obtain that the expected proportions of the d distinct elements that are reconciled in the first, second, third, and fourth round are 0.962, 0.0380, 3.61×10^{-4}, and 2.86×10^{-6} respectively under the optimal parameter settings ($n = 127$, $t = 13$) for the instance used twice above (with $d = 1{,}000$, $r = 3$, $\delta = 5$ and $p_0 = 0.99$). This confirms our earlier claim that the vast majority

(> 95%) of the distinct elements are reconciled, and hence most of the communication overhead is incurred, in the first round.

6 ESTIMATE d

We have so far assumed that d is precisely known. In reality, d is not known a priori in most applications. In this case, Alice and Bob need to first obtain a relatively accurate estimate of d. To this end, we propose a new set difference cardinality estimator that is based on the celebrated Tug-of-War (ToW) sketch [3].

6.1 The ToW Estimator

The ToW sketch was originally proposed in [3] for a subtly related but very different application: to estimate F_2, the second frequency moment of a data stream. We discover that ToW can also be used for estimating the set difference cardinality d as follows. Given a universe $\mathcal{U}$, let $\mathcal{F}$ be a family of four-wise independent hash functions, each of which maps elements in $\mathcal{U}$ to +1 or −1 each with probability 0.5. The ToW sketch of a set $S \subset \mathcal{U}$, generated using a hash function $f \in \mathcal{F}$, is defined as $Y_f(S) \triangleq \sum_{s \in S} f(s)$, the sum of the hash values of all elements in S. Using the same analysis derived in [3], we can prove that $\hat{d} = \big(Y_f(A) - Y_f(B)\big)^2$ is an unbiased estimator for $d = |A \triangle B|$, as long as f is drawn uniformly at random from $\mathcal{F}$. The variance of this estimate is $(2d^2 - 2d)$. The proof for the unbiasedness of this estimator and the calculation for its variance can be found in Appendix A in [12]. For notational convenience, we drop the subscript f from Y_f and add a different subscript to Y in the sequel.

The estimate obtained from a single sketch is usually not very accurate. To achieve high estimation accuracy, multiple sketches, generated using independent hash functions, can be used. Suppose ℓ such sketches, which we name Y_1, Y_2, ..., Y_ℓ, are used. The ToW estimator using these ℓ sketches is given by $\hat{d} = \big(\sum_{i=1}^{\ell}(Y_i(A) - Y_i(B))^2\big)/\ell$. The variance of $\hat{d}$ is $(2d^2 - 2d)/\ell$, which is ℓ times smaller than if only a single ToW sketch is used.

Space Complexity. Each ToW sketch for any set S is an integer within the range $[-|S|, |S|]$, and is hence at most $\log(2|S| + 1)$ bits long. Therefore, the space complexity of the ToW estimator using ℓ sketches is $\ell \cdot \log(2|S| + 1)$ bits. We use ℓ=128 totaling 336 bytes in PBS to achieve an appropriate level of estimation accuracy that we will elaborate next.

6.2 Use The ToW Estimator in PBS

The ToW estimator is to be used by PBS, or by any other set reconciliation algorithm that needs this step, at the very beginning (before the reconciliation process starts), as follows. Alice sends the $\ell = 128$ ToW sketches of set A to Bob. Upon receiving these ℓ ToW sketches, Bob computes $\hat{d}$ as shown above and sends $\hat{d}$ to Alice. Both Alice and Bob then conservatively assume that the actual d is $1.38\hat{d}$ and compute the optimal n and t values (described in §5.1) accordingly. We use $\gamma = 1.38$ here, because it is found (through Monte-Carlo simulations) to be the smallest γ value to guarantee that $Pr[d \le \gamma\hat{d}] \ge 99\%$ for the ToW estimator using 128 sketches. Using more (than 128) sketches allows γ to be smaller, but (128, 1.38) appears to strike a nice tradeoff according to our simulations.

In our evaluation to be described in §8, we assume that d is not known a priori. Like in PBS, we use the ToW estimator with 128 sketches with a total cost of 336 bytes also for two of our "competitors" PinSketch and D.Digest, because, as have been explained in Appendix B in [12], the ToW estimator is the most space-efficient among all existing estimators. In calculating the communication overheads of all three of them (PBS, PinSketch, D.Digest), this overhead of 336 bytes is excluded. For a fair comparison, we subtract this amount (336 bytes) from the communication overhead of another competitor Graphene, as Graphene does not require an estimator.

7 RELATED WORK

In this section, we provide a brief survey of existing set reconciliation algorithms. As mentioned in §1.1, in describing and comparing them with PBS, we only consider the unidirectional set reconciliation in which Alice learns $A \triangle B$.

Bloom filters (BF) [4] can be used to construct a crude set reconciliation scheme as follows. First, Alice and Bob exchange BFs for sets A and B. Upon receiving the BF (for B) from Bob, Alice can obtain an estimate, denoted as $\widehat{A \setminus B}$, of the set $A \setminus B$ by checking each element in A against this BF. Note that $\widehat{A \setminus B}$ is in general an underestimate: $\widehat{A \setminus B}$ may not contain all elements in $A \setminus B$, because this BF may produce false positives that each suggests an element is in B when it is not. Similarly, Bob can obtain $\widehat{B \setminus A}$ and sends it to Alice, from which Alice can infer an underestimate (of $A \triangle B$) $\widehat{A \triangle B} = \widehat{A \setminus B} \bigcup \widehat{B \setminus A}$. Set reconciliation solutions that build on and extend this BF-based technique, including [6, 14, 19], all suffer from this underestimation problem, and hence are only suitable for few applications that do not require perfect data synchronization.

As mentioned earlier, most exact set reconciliation algorithms are based on either invertible Bloom filters (IBF) [13] or error-correction codes (ECC).

IBF-Based Algorithms. In an IBF, each element (from a set) is inserted into k cells indexed by k independent hash functions. Whereas each cell is a single bit in a BF, it has three fields in an IBF, each of which requires a single word of length $\log |\mathcal{U}|$. Therefore, IBFs are much more powerful than BFs: The set difference $A \triangle B$ of sets A and B can be recovered from the "difference" of their IBFs using a "peeling process" similar to that used in the decoding algorithms for some erasure-correcting codes, such as Tornado codes [18]. For this decoding process to succeed with a high enough probability, IBF-based solutions, such as Difference Digest (D.Digest) [11], have to use roughly $2d$ cells. This translates into a communication overhead of roughly $6d \log |\mathcal{U}|$, or 6 times the theoretical minimum.

A recent solution called Graphene [25] reduces the high communication overhead of IBF-based solutions by augmenting it with BFs. Here, we only describe its simplest version (Protocol I in [25]) that works only for the special case of $B \subset A$. Its basic idea is for Alice to first obtain $\widehat{A \setminus B}$, an underestimate of of $A \setminus B$, by querying the BF for the set B as described above, and then recover only the "missing" part $(A \setminus B) \setminus (\widehat{A \setminus B})$ using an IBF. When the BF is configured to have a reasonably low false positive rate say ϵ, the IBF needs only to "encode" the roughly ϵd "missing" distinct elements rather than all the d elements in $A \setminus B$, resulting in savings

of "$O((1-\epsilon)d)$" in the size of IBF. In general, for this ϵ to be reasonably low (say meaningfully away from 1), the size of the BF has to be $O(|B|)$ with a nontrivial constant factor [4]. However, when $|B| \gg d$, which as explained earlier is often the case in most applications, the savings of "$O((1-\epsilon)d)$" in the IBF size is no longer worth the $O(|B|)$ cost of the BF; in this case Graphene drops the BF and degenerates to an IBF-only solution. For this reason, Graphene is more communication-efficient than other IBF-based solutions only when d is sufficiently large with respect to $|B|$. Furthermore, this efficiency grows with d, as we will show in §8.2.

ECC-Based Algorithms. The basic ideas of ECC-based algorithms [10, 15, 22, 24] are similar to that of PinSketch [10]. Given a universe $\mathcal{U}$ in which each element is assigned an index between 1 and $|\mathcal{U}|$, PinSketch encodes each set $S \subset \mathcal{U}$ as a $|\mathcal{U}|$-bit-long bitmap $S[1..|\mathcal{U}|]$: $S[i]$ (the i^{th} bit of $S[1..|\mathcal{U}|]$) is equal to 1 if the element, whose index is i, is contained in S; otherwise, $S[i] = 0$. In contrast, in PBS the size n of a bitmap depends only on d (in PBS-for-small-d) or δ (in PBS-for-large-d), and not on the size of the universe or the cardinality of the group the bitmap encodes, and is hence much shorter.

In PinSketch the d distinct elements in $A \triangle B$ are "indexed" by the d bit positions in which the two $|\mathcal{U}|$-bit-long bitmaps encoding A and B respectively differ. Like in PBS, these d bit locations can be learned by letting Alice send BoB a BCH codeword encoding A's bitmap. However, whereas the length of BCH codeword in PBS is $d \log n$ or $\log n$ per distinct element, that in PinSketch is $d \log |\mathcal{U}|$, or $\log |\mathcal{U}|$ per distinct element. Hence, the BCH codeword is typically 3 to 4 times longer ($\log |\mathcal{U}| = 32$ bits in the example above) in PinSketch than in PBS ($\log n = 8$ bits in the example above), a fact we will use in §8.3.

As mentioned earlier, ECC-based algorithms suffer from a much higher decoding computational complexity of at least $O(d^2)$. In [21], a partition-based solution was proposed to reduce this computational complexity to $O(d)$, but in a different manner than the partitioning in PBS. This solution requires $O(\log d)$ rounds of message exchanges, which is generally much larger than that in PBS.

8 PERFORMANCE EVALUATION

In this section, we evaluate the performance of PBS, and compare it against three state-of-the-art algorithms that we have described in detail in §7: PinSketch [10], Difference Digest (D.Digest) [11], and Graphene [25]. In §8.3, we apply the partitioning technique used in PBS to PinSketch to reduce its decoding computational complexity and compare PBS against it. Our evaluation is mainly focused on two performance metrics: *communication overhead* and *computational overhead*. The former is measured by the total amount of data transmitted between Alice and Bob to allow Alice to learn $A \triangle B$. The latter includes both encoding and decoding times.

The evaluation shows conclusively that PBS strikes a much better tradeoff between communication and computational overheads than all three algorithms. It has a communication overhead much lower than IBF-based techniques such as D.Digest and Graphene, and only slightly higher than PinSketch, whose computational overhead is much larger. In addition, PBS has the lowest computational overhead among all four algorithms.

Experiment Setup. Our evaluation uses a key space (universe) $\mathcal{U}$ of all 32-bit binary strings. In other words, the (hash) signature length is $\log |\mathcal{U}| = 32$. Like in [11], all set pairs are created as follows. First, elements in A are drawn from $\mathcal{U}$ uniformly at random without replacement. Then, precisely $|A| - d$ elements in A are sampled also uniformly at random without replacement to make up set B, so that the set difference $A \triangle B$ contains exactly d elements.

In all experiments, we fix the cardinality of A at 10^6 and let the value of d vary from 10 to 10^5. For each value of d, we create a set of 1,000 mutually independent instances of (A, B). Each point in each plot is the average of 1,000 experimental results on such a set of 1,000 instances. All experiments were performed on a workstation with an Intel Core i7-9800X processor running Ubuntu 18.0.4.

Implementations. We implement PBS in C++. We use the `xxHash` library [8] for generating all hash functions in PBS, including those in the ToW estimator. The Minisketch library [26], released by the authors of [24], is used for the BCH encoding and decoding in both PBS and PinSketch [10]. As the authors of D.Digest [11] have not released their source code, we implement it using the open-source code of IBFs in C++ released by the authors of [25]. For evaluating Graphene [25] fairly, we have made the following revision to the source code provided by the authors of [25] to make it as computationally efficient as possible. The original source code was written in Python, with the most computationally expensive part implemented in C++ with a Python wrapper. We have rewritten all Python code of it in C++.

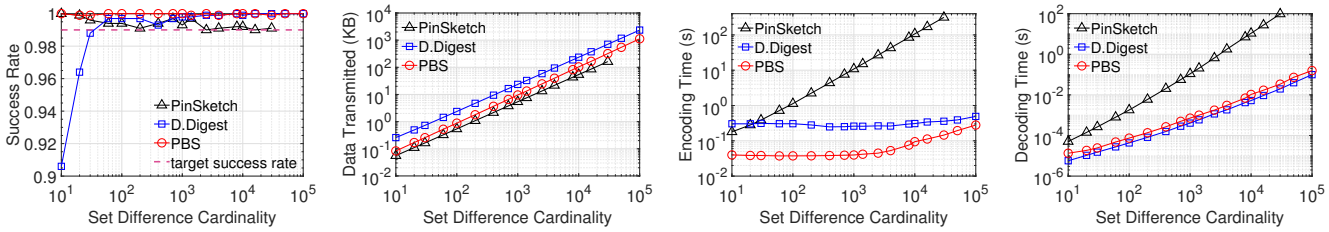
8.1 PBS vs. PinSketch and D.Digest

In this section, we compare PBS with PinSketch and D.Digest. We keep the comparison of PBS with Graphene separate in §8.2, because a fair comparison there calls for slightly different experimental settings and parameters.

8.1.1 Parameter configurations. As explained earlier, in virtually all applications, a set reconciliation algorithm should guarantee a high enough *success rate (probability)* of reconciling all distinct elements in $A \triangle B$, and guaranteeing a higher success rate generally requires higher communication and computational overheads. Hence, to fairly compare these set reconciliation algorithms, we should properly configure their parameters so that they roughly have the same success rate. In [11], the authors have provided configuration guidelines for tuning D.Digest to achieve a success rate of 0.99. To tune the parameters of D.Digest to achieve other success rates, however, requires a large number of Monte-Carlo experiments. Instead, we tune the parameters of PinSketch and PBS to match this success rate of D.Digest, because it is much easier to do so for PinSketch (to be shown next) and PBS (shown in §5.1).

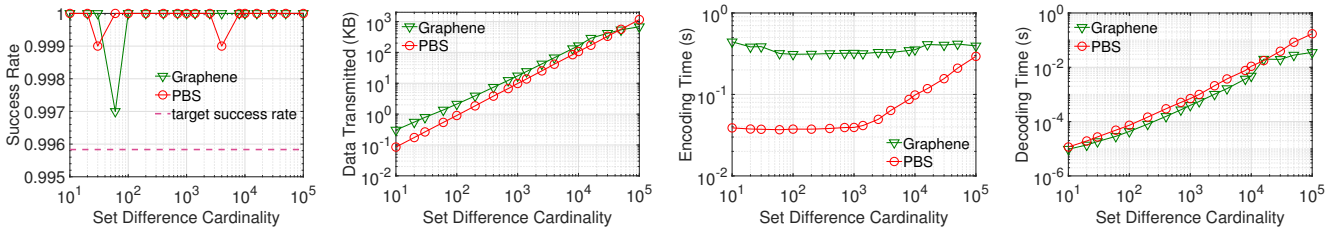
PinSketch. As explained earlier in §6, $Pr[d \le 1.38\hat{d}] \ge 0.99$, when $\hat{d}$ is obtained from the ToW estimator with 128 sketches. We set the BCH error-correction capacity t to $1.38\hat{d}$ so that the event $\{d \le t\}$ which corresponds to successful BCH decoding and hence set reconciliation, has a probability of at least 0.99.

D.Digest. As suggested in [11], we use $2\hat{d}$ cells (to both account for the randomness of $\hat{d}$ and allow accurate IBF decoding) in the IBF of D.Digest, and use 3 hash functions if $\hat{d}$ is greater than 200 and 4 hash functions otherwise.



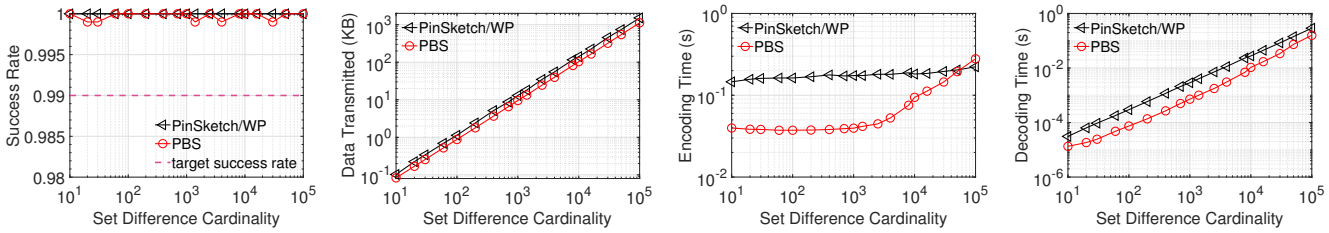
(a) Success rate **(b) Communication overhead** **(c) Encoding time** **(d) Decoding time**

Figure 1: Comparisons against PinSketch and D.Digest, with a target success rate of 0.99.



(a) Success rate **(b) Communication overhead** **(c) Encoding time** **(d) Decoding time**

Figure 2: Comparisons against Graphene, with a target success rate of 239/240.



(a) Success rate **(b) Communication overhead** **(c) Encoding time** **(d) Decoding time**

Figure 3: Comparisons against PinSketch/WP, with a target success rate of 0.99.

PBS. We choose $r = 3$ rounds since it is a sweet spot as explained in §5.2. We set the target success probability p_0 to 0.99 and optimally parameterize PBS using the procedure described in §5.1. In each experiment, we allow PBS to run at most 3 rounds and its communication and computational overheads are measured as the total during all rounds executed.

8.1.2 Experimental results. We report the experimental results in this section. Note that we were not able obtain the results for PinSketch for $d > 30{,}000$ in a reasonable amount of time, as it is prohibitively expensive computationally to do so. We first present the success rates of all three algorithms in Figure 1a, which shows that all these algorithms have achieved success rates at least as high as the target success rate (0.99) under all different values of d except for D.Digest whose success rates are slightly lower than 0.99 when $d \leq 30$.

Communication Overhead. Figure 1b compares the communication overhead of PBS against those of PinSketch and D.Digest. Results show that the communication overheads scale approximately linearly with respect to d for all three algorithms. More precisely, for any d, the amount of communication per distinct element is roughly a constant. D.Digest is the worst. It requires around 6×32 bits per distinct element, 6 times the theoretical minimum (32 bits per distinct element). PBS is much better, the communication overhead of which is between 2.13 to 2.87 times the theoretical minimum. PinSketch has the lowest communication overhead, which is 1.38 times the theoretical minimum.

Encoding Time. Figure 1c compares the encoding time of PBS against those of PinSketch and D.Digest. Figure 1c clearly shows that the former is much lower than the latter under all different values of d.

Decoding Time. Figure 1d compares the decoding time of PBS against those of PinSketch and D.Digest. As shown in Figure 1d, the decoding time of PinSketch is much higher than those of D.Digest and PBS when d is large (say $\geq$ 1,000). Figure 1d also shows clearly that D.Digest is the best, whose decoding time is 1.53 to 2.35 times shorter than that of PBS.

However, as discussed earlier, the encoding time of D.Digest is much (up to one order of magnitude) longer than that of PBS, and encoding time (of PBS and D.Digest) is usually much longer than

the corresponding decoding time. Thus, PBS has the lowest overall computational overhead.

8.2 PBS vs. Graphene

In this section, we compare PBS against Graphene. Recall that in our experimental setting, we have $B \subset A$ and need to let Alice learn $A \Delta B$, which was shown in [25] to be the best-case scenario for Graphene in terms of communication overhead and decoding time. Hence we have treated Graphene more than fairly here. Since the parameters in the source code provided by the authors of [25] are already optimized for achieving a target success rate of 239/240, we tune PBS to match this success rate: The success rates of both PBS and Graphene are higher than 239/240, as shown in Figure 2a.

Communication Overhead. Figure 2b compares the communication overhead of PBS against that of Graphene. It shows that, even in this best-case scenario for Graphene, PBS has much lower (roughly 1.2 to 7.4 times less) communication overhead than Graphene under all different values of d except when d gets very close to 100,000. The reason behind this exception was explained earlier (in §7): When d is sufficiently large with respect to $|A|$ ($=10^6$ in this case), it becomes more communication-efficient overall for Graphene to start using a BF to reduce the size of its IBF. It can be calculated using an optimization formula in [25] that the breakeven point (for using a BF) in this case is some number between $d = 10{,}000$ and $d = 16{,}000$. We can actually see in Figure 2b that the slope of the Graphene curve, which corresponds to the average communication overhead per distinct element, starts to decrease after the breakeven point, resulting in it eventually going under the PBS curve roughly after $d \geq 50{,}000$.

Encoding Time. Figure 2c clearly shows that the encoding time of PBS is 1.34 to 11.38 times lower than that of Graphene under all values of d.

Decoding Time. Figure 2d compares the decoding time of PBS against those of Graphene and clearly shows that the former is slightly (1.20 to 2.28 times) longer than the latter except when d is close to 100,000 where the former is up to 4.87 times longer.

8.3 PBS vs. PinSketch with Partition

Arguably, the same algorithmic trick (*i.e.*, hash-partition A and B each into groups) can be applied also to PinSketch [10] for reducing its BCH decoding time from $O(d^2)$ to $O(d)$. Doing so however makes the communication overhead of PinSketch higher than that of PBS for the following reason. As explained in §3.1, we need to leave a safety margin in setting the BCH error-correction capacity t, in the sense that t needs to be "comfortably" larger than δ, the average number of "bit errors" per group. Hence for each group pair, the average additional communication overhead (incurred for transmitting a longer BCH codeword) of leaving this safety margin is $(t-\delta)\log n$ in PBS and is $(t-\delta)\log|\mathcal{U}|$ in PinSketch. However, as explained in §7, $\log n$ is typically 3 to 4 times smaller than $\log|\mathcal{U}|$. Hence PinSketch pays 3 to 4 times more for leaving the safety margin, resulting in a higher overall communication overhead, as will be elaborated next.

Now we compare the performance of PBS against that of PinSketch with hash partition, which we refer to as PinSketch/WP. For PinSketch/WP, we use the same δ and t values as in PBS (there is no parameter n in PinSketch/WP since it does not use a parity bitmap), with a target success probability of p_0=0.99 within r=3 rounds, in each experiment instance. The experimental results are reported in Figure 3. It clearly shows that PBS outperforms PinSketch/WP in both communication overhead and computational overhead (the sum of the encoding and the decoding time). Note this outperformance will increase when the hash signature length $\log|\mathcal{U}|$ increases ($\log|\mathcal{U}|=32$ bits in Figure 3). Hence, PBS would outperform by a wider margin in real-world blockchain applications where $\log|\mathcal{U}|$ is much larger (e.g., $\log|\mathcal{U}|=256$ bits in Bitcoin [23]), as has been shown in Appendix I.2 in [12].

8.4 Number of Rounds Required by PBS

Table 2: Number of rounds r needed for full reconciliation.

d \ r	1	2	3
10	0.804	0.188	0.008
100	0.217	0.760	0.023
1,000	0	0.957	0.043
10,000	0	0.907	0.093
100,000	0	0.818	0.182

In this section, we investigate the empirical number of rounds r required by PBS to correctly reconcile all d distinct elements. The parameter settings are exactly the same as those we used in §8.1. The only difference is that here we let PBS run as many rounds as needed instead of at most 3 rounds. Table 2 presents the empirical distributions of the number of rounds required by PBS to correctly reconcile all d distinct elements, with d=10, 100, 1,000, 10,000 and 100,000. The average numbers of rounds needed are 1.20, 1.81, 2.04, 2.09 and 2.18 for d=10, 100, 1,000, 10,000 and 100,000 respectively. Furthermore, in every experiment the reconciliation process took no more than 3 rounds to complete. Hence the 3 probability values in every row of Table 2 add up to (probability) 1.

9 CONCLUSION

In this paper, we propose Parity Bitmap Sketch (PBS), a space- and computationally-efficient solution to the set reconciliation problem. We show, through experiments, that PBS strikes a much better tradeoff between communication and computational overheads than all the state-of-the-art solutions. In addition, we derive a novel rigorous analytical framework for PBS, which most existing solutions do not have. Through three applications of this framework, we demonstrate that it enables both the accurate analysis of various performance metrics and the tuning of the parameters of PBS for near-optimal performances.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048 and CNS-2007006. We thank the reviewers and the shepherd for their critiques, comments and suggestions that have greatly improved the quality and the readability of this paper.

REFERENCES

[1] [n.d.]. Dropbox Smart Sync. https://www.dropbox.com/smart-sync. [Online; accessed 23-July-2020].

[2] [n.d.]. Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.org/. , 32 pages. [Online; accessed 10-July-2020].

[3] Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The Space Complexity of Approximating the Frequency Moments. *J. Comput. System Sci.* 58, 1 (1999), 137–147.

[4] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422–426. https://doi.org/10.1145/362686.362692

[5] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. 1960. On A Class of Error Correcting Binary Group Codes. *Information and Control* 3, 1 (1960), 68–79.

[6] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. 2004. Informed Content Delivery across Adaptive Overlay Networks. *IEEE/ACM Transactions on Networking* 12, 5 (Oct. 2004), 767–780. https://doi.org/10.1109/TNET.2004.836103

[7] Dwaine Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. 2003. Incremental Multiset Hash Functions and Their Application to Memory Integrity Checking. In *Advances in Cryptology - ASIACRYPT 2003*. Berlin, Heidelberg, 188–207.

[8] Yann Collet. [n.d.]. xxHash - Extremely fast hash algorithm. https://github.com/Cyan4973/xxHash.

[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems* 31, 3, Article 8 (Aug. 2013), 22 pages. https://doi.org/10.1145/2491245

[10] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. 2008. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM J. Comput.* 38, 1 (2008), 97–139.

[11] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. 2011. What's the Difference? Efficient Set Reconciliation Without Prior Context. In *Proceedings of the ACM Special Interest Group on Data Communication* (Toronto, Ontario, Canada). 218–229. https://doi.org/10.1145/2018436.2018462

[12] Long Gong, Ziheng Liu, Liang Liu, Jun Xu, Mitsunori Ogihara, and Tong Yang. 2020. Space- and Computationally-Efficient Set Reconciliation via Parity Bitmap Sketch (PBS). arXiv e-prints, arXiv:2007.14569.

[13] Michael T. Goodrich and Michael Mitzenmacher. 2011. Invertible Bloom Lookup Tables. arXiv e-prints, arXiv:1101.2245. arXiv:1101.2245 http://arxiv.org/abs/1101.2245

[14] Deke Guo and Mo Li. 2013. Set Reconciliation via Counting Bloom Filters. *IEEE Transactions on Knowledge and Data Engineering* 25, 10 (Oct. 2013), 2367–2380. https://doi.org/10.1109/TKDE.2012.215

[15] M. G. Karpovsky, L. B. Levitin, and A. Trachtenberg. 2003. Data verification and reconciliation with generalized error-control codes. *IEEE Transactions on Information Theory* 49, 7 (July 2003), 1788–1793. https://doi.org/10.1109/TIT.2003.813498

[16] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review* 44, 2 (April 2010), 35–40. https://doi.org/10.1145/1773912.1773922

[17] Norman Levinson. 1946. The Wiener (root mean square) error criterion in filter design and prediction. *Journal of Mathematics and Physics* 25, 1-4 (1946), 261–278.

[18] Michael Luby. 1998. Tornado codes: Practical erasure codes based on random irregular graphs. In *Proceedings of International Workshop on Randomization and Approximation Techniques in Computer Science*. 171–171.

[19] Lailong Luo, Deke Guo, Ori Rottenstreich, Richard T.B Ma, and Xueshan Luo. 2019. Set Reconciliation with Cuckoo Filters. In *Proceedings of the ACM International Conference on Information and Knowledge Management* (Beijing, China). 2465–2468. https://doi.org/10.1145/3357384.3358065

[20] Ralph C. Merkle. 1987. A Digital Signature Based on a Conventional Encryption Function. In *Proceedings of the Conference on The Theory and Application of Cryptographic Techniques*. Berlin, Heidelberg, 369–378.

[21] Yaron Minsky and Ari Trachtenberg. 2002. Practical set reconciliation. In *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*, Vol. 248.

[22] Y. Minsky, A. Trachtenberg, and R. Zippel. 2003. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory* 49, 9 (Sept. 2003), 2213–2218. https://doi.org/10.1109/TIT.2003.815784

[23] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system.* Technical Report. Manubot. https://bitcoin.org/bitcoin.pdf.

[24] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. 2019. Erlay: Efficient Transaction Relay for Bitcoin. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom). 817–831. https://doi.org/10.1145/3319535.3354237

[25] A. Pinar Ozisik, Gavin Andresen, Brian N. Levine, Darren Tapp, George Bissias, and Sunny Katkuri. 2019. Graphene: Efficient Interactive Set Reconciliation Applied to Blockchain Propagation. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China). 303–317. https://doi.org/10.1145/3341302.3342082

[26] Pieter Wuille. [n.d.]. Minisketch: an optimized library for BCH-based set reconciliation. https://github.com/sipa/minisketch.