GenSync: A New Framework for Benchmarking and Optimizing Reconciliation of Data

Novak Boškov, Ari Trachtenberg, David Starobinski

***Abstract*—In the *set reconciliation* problem, remote parties seek to reconcile similar sets of data according to an efficiency objective, such as minimizing communication or computation. Though investigated for many individual distributed applications, this problem still lacks a holistic treatment, and this is the aim of this work. Specifically, we design and analyze *GenSync*, a unified set reconciliation framework that incorporates several state-of-the-art set reconciliation protocols with an integrated testbed. We compare and analyze the various protocols and offer general guidelines for selecting a good protocol for a given application. Through extensive experiments, we demonstrate that the optimal choice of protocol is highly sensitive to several parameters, including network properties (*e.g.,* bandwidth and latency) and computing power. Notably, none of our framework's protocols are universally dominant under diverse conditions. To demonstrate our framework, we measure the effects of protocol choice in reconciling memory pools of adjacent Bitcoin nodes.**

***Index Terms*—Peer-to-peer computing, Data communication, Approximate computing**

I. Introduction

Set reconciliation is a building block of many disparate distributed systems. For instance, recently proposed transaction and block dissemination protocols of popular cryptocurrencies rely on set reconciliation as the means of tackling scaling issues [1]–[3]. On the other hand, information systems, such as distributed databases [4]–[8] and distributed file systems [9], utilize reconciliation for loosely consistent replica synchronization. In collaborative editing, set reconciliation is used to support decentralization [10], [11] and maximize performance on mobile devices [12]. Even wireless sensor networks (WSNs) apply reconciliation to reduce the communication overhead among sensors [13].

In the classical scenario, the set reconciliation problem involves two remote parties, Alice and Bob, with their data sets S_A and S_B, respectively. When Alice seeks to learn the elements of $S_B \setminus S_A$ (*i.e.,* the elements in S_B but not in S_A), we use the term *one-way* set reconciliation [11], [14]–[16]. If in addition, Bob seeks to learn $S_A \setminus S_B$, we call this process a *full* set reconciliation. In the latter case, Alice and Bob can deduce the *symmetric difference*, which is defined as follows

$$S_A \oplus S_B = (S_A \setminus S_B) \cup (S_B \setminus S_A).$$

Over the past years, the set reconciliation problem has been actively studied and a multitude of protocols have been proposed [11], [17]–[26], largely falling into two camps. The first camp makes use of carefully gauged approximate set membership data structures (ASM) such as Bloom filters (BF) [13], [26], invertible Bloom lookup tables (IBLT) [19], cuckoo filters [18], [21]. The second camp makes use of a fundamental connection between reconciliation and error correcting codes [14], applying, for example, Bose-Chaudhuri-Hocquenghem (BCH) [24] and Reed-Solomon (RS) codes [11], [20], [27] to the reconciliation problem.

Although the common idea permeating all the proposed protocols is to trade-off a little accuracy in favor of efficiency, there are significant performance differences among the protocols, depending on the application. Indeed, the performance of each protocol depends on a number of factors, ranging from network conditions to the computational capabilities of the nodes. While some protocols may perform exceptionally well under conditions of low latency and high bandwidth, others may dominate under a constrained bandwidth budget. As a result, it is currently challenging to reason about the best reconciliation protocol for a given application.

Contributions

In this work, we study the relative efficiency of several popular set reconciliation protocols through a novel framework we call *GenSync*.

More specifically, these main contributions herein include:

- We introduce *GenSync*, the first extensible public framework that unifies popular data reconciliation algorithms under a common programming interface [28].
- We contribute the *GenSync testbed*, a novel benchmarking tool for experimental comparison of data reconciliation algorithms.
- We offer experimental and analytical comparisons of the set reconciliation algorithms that are part of *GenSync*, and categorize situations where each excels or lags.
- As a proof of concept, we evaluate the performance of the *GenSync*'s algorithms for Bitcoin mempool reconciliation using data from two live Bitcoin full nodes.

A key finding is that there is no *universally dominant* set reconciliation protocol among those we examined. Although IBLT-based set reconciliation dominates when there is sufficient bandwidth and symmetric computational power, cuckoo-based methods dominate under asymmetric computation. Yet, if bandwidth is the bottleneck, coding-inspired approaches prevail. Our findings come from a novel approach to comparing set reconciliation protocols, by introducing a metric called *total time to reconcile* (TTR). Unlike previous metrics, TTR

This research was supported in part by NSF grant CCF-1563753.

The authors are with the ECE Department, Boston University, Boston, MA, 02215, USA. (email: {boskov,trachten,staro}@bu.edu).

Author	Algorithm	Applicability	Class
Skjegstad and Torleiv [29]	BF	General	ASM
Tian et al. [30]			
Fan et al. [26]	Counting BF		
Luo et al. [21]	CF		
Li et al. [18]	Counting CF		
Luo et al. [31]	Marked CF		
Eppstein and Goodrich [19]	IBLT		
Ozisik et al. [1]	BF + IBLT	Blockchain	
Minsky and Trachtenberg [27]	CPI	General	ECC
Dodis et al. [24]	BCH		
Gong et al. [23]	Parity Bitmap Sketch		
Jin et al. [20]	CPI	Disruption Tolerant Networks	
Naumenko et al. [2]	BCH + low-fanout flooding	Blockchain	

TABLE I: Set reconciliation literature summary. **BF** *abbr.* Bloom filter. **CF** *abbr.* Cuckoo filter. **IBLT** *abbr.* Invertible Bloom Lookup Table. **ASM** *abbr.* Approximate Set Membership. **CPI** *abbr.* Characteristic Polynomial Interpolation. **BCH** *abbr.* Bose-Chaudhuri-Hocquenghem (codes). **ECC** *abbr.* Error-Correcting Codes.

factors in key system parameters, such as network conditions and the amount of available compute at the peers. Based on the concept of TTR, the *GenSync* testbed allows for quantification of the performance differences among the protocols for the given system characteristics.

Roadmap

The rest of this paper is structured as follows. We review related work and provide background on reconciliation algorithms that form the backbone of *GenSync* in Section II. In Section III, we introduce *GenSync*, including its abstractions and design choices that underscore its programming interface. Section IV analyzes the properties of algorithms included in the *GenSync* framework. We set forth the results of our experiments in Section V and focus on the benefits of interactive protocols in Section V-E. In Section VI, we present a proof of concept application of our framework to synchronize Bitcoin mempool data sets. We conclude in Section VII.

II. Background And Related Work

We next present an overview of the set reconciliation literature, summarized in Table I. As mentioned in the introduction, there are two broad families of set reconciliation techniques:

1) *Approximate set membership data structures-based* (ASM), and
2) *Error-correcting codes-based* (ECC).

The techniques differ in terms of their main algorithmic ideas and applicability. While some approaches apply generally to the set reconciliation problem presented in the introduction, others are developed for specialized purposes such as block propagation in blockchains or data reconciliation in disruption tolerant networks. Later, in Section III, we discuss the concrete implementations of several of these techniques in the *GenSync* framework. We then compare them analytically in Section IV.

In the rest of this section we focus on the major algorithmic ideas that influenced set reconciliation approaches. Throughout this work we use

$$d = |S_A \oplus S_B|$$

to denote the *size of symmetric difference* between the sets S_A and S_B.

A. Bloom Filters (BF)

Approximate set membership (ASM) data structures provide an approximate set representation typically with constant access time and sub-linear memory. A common example of an ASM data structure is the Bloom filter (BF) [32], [33], which utilizes k different and independent hash functions to hash each element $s \in S$ in the set. The filter uses the hash values to set the corresponding bits in the underlying bit vector. The lookup operation follows the same procedure and answers positively when any of the bits that k hash functions point to is set. To achieve some given *false positive* rate of its lookup queries (ε), Bloom filter uses $k = \log\left(\frac{1}{\varepsilon}\right)$ independent hash functions resulting in $\frac{1}{\ln 2}k$ bits per item in S.

One way to utilize Bloom filters in a set reconciliation algorithm would be for Alice to encode her set S_A into a Bloom Filter and to send it to Bob who can then query his elements S_B to learn $S_B \setminus S_A$, up to the false positive rate ε. The lookup queries on Bob's elements that are not in S_A will come out negative while certain amount of Bob's elements that are not present in S_A will come out falsely positive (up to ε). Once Bob knows which of his S_B elements are unique to his set, he can hand them over to Alice. On a high level, many ASM-based set reconciliation algorithms follow the structure of the Bloom filter-based protocol. However, since the size of a Bloom filter is linear in the size of the set it encodes, sending a Bloom filter to the peer would not be communication-efficient when the actual size of differences d is small relative to the sizes of the sets. Also note that Bob needs to pass all his elements through Alice's Bloom filter because Bloom filters do not support element listing that would allow Bob to directly learn Alice's set. In the rest of this section, we give a brief literature overview of the similar ASM-based set reconciliation protocols that do better than the previously described Bloom filter-based protocol.

B. Counting Bloom Filter (CBF)

Fan et al. [25] introduced counting Bloom filter (CBF) as an alternative to traditional Bloom filter when the deletion operation is needed. Instead of using bitvector as the underlying

data structure of the traditional Bloom filters, CBF utilizes a vector of counters such that insertions increase the counter, while deletions decrease the counter. The lookup queries are similar to those in the traditional Bloom filters except that now the presence of an element in the cell is signalized by a non-zero value in the corresponding counter cell. Guo and Li [26] use CBF's to implement a set reconciliation protocol. For this purpose, they define the subtraction and the union operations between the two equally long CBF_A and CBF_B. They define CBF-wise subtraction as the subtraction of the counter values in CBF_A by the counter values in CBF_B for each cell. Effectively, the cell pairs corresponding to the common keys between CBF_A and CBF_B will cancel out leaving only the keys corresponding to the elements that are either unique to Alice or Bob. Analogously, they define the CBF-wise union as the summation of the counter values from CBF_A and the corresponding counter values in CBF_B for each cell. Using such defined subtraction and union operations, Guo et al. derive the following approximation [26]:

$$CBF_A - CBF_B \approx CBF(A \setminus B) - CBF(B \setminus A) \quad (1)$$

Thus, each party can approximately learn its local elements by subtracting its own CBF by the one received from the peer and then passing its own elements through the resulting filter.

C. Invertible Bloom Filter (IBF)

Eppstein and Goodrich [34] introduce IBF as a novel data structure with the following two distinctive features over traditional Bloom filters: 1) decoding ("peeling"); the process that allows for listing all the elements in an IBF, and 2) subtraction; the operation that allows for subtracting two IBFs to obtain $S_A \setminus S_B$ directly. Utilizing the capabilities of IBF, Eppstein and Goodrich define their set reconciliation protocol as follows. First, Alice encodes her S_A into IBF_A that can decode up to d elements and sends it to Bob. On the other side of the communication channel, Bob encodes his own elements S_B into IBF_B and applies $IBF_B - IBF_A$, where the minus sign denotes the IBF subtraction. With certain probability (see Section IV-C), Bob will succeed to decode the resulting IBF which yields $S_B \setminus S_A$ directly.

Goodrich and Mitzenmacher [35] extend invertible Bloom filter data structure to a lookup table that they dubbed invertible Bloom lookup table (IBLT). Their data structure has a design parameter t; the threshold on maximal number of key-value pairs that the data structure can decode (list). Thus, IBLT uses the space at most linear in t. As long as an IBLT contains less than t key-value pairs, listing operation succeeds with a high probability. As in Bloom filter and IBF, IBLT lookup takes $O(k)$, while listing takes $O(t)$ time. Importantly, Goodrich and Mitzenmacher [35] prove that IBLT's can list the count of identical key-value pairs that reside in the data structure. This feature is especially helpful when S_A and S_B that we need to reconcile are multisets instead of sets.

D. Cuckoo Filter (CF)

Cuckoo filter [36]–[38] is an ASM data structure based on cuckoo hashing [39]. As opposed to traditional Bloom filters and their variations, cuckoo filters represent keys using their *fingerprints*. A key fingerprint is often a hash of significantly shorter length than the key itself and each fingerprint is stored in its own slot of the cuckoo filter. A small fixed number of slots is then organized into a number of buckets. During the key lookup, we rely on a hash function that ranges over the number of buckets to obtain the first *candidate bucket* for the key. Additionally, we utilize the technique called *partial-key cuckoo hashing* to obtain the other candidate bucket. Once the two candidate buckets are known, we linearly scan each of them and return a positive answer when any of the buckets contain the fingerprint of the given key. On the other hand, the insertion process is more complex than in the case of Bloom filter and its variants. In particular, we need to take care of bucket overflows and possibly evict certain number of fingerprints to their *alternative buckets*. This process is recursive and may fail with some small probability. However, for a number of practical false positive rates (ε), cuckoo filters are more space efficient than Bloom filter. Relying on this distinctive feature of cuckoo filters, Luo et al. [21] proposed a set reconciliation protocol. Their reconciliation protocol is similar to the traditional Bloom filter-based one in so that Alice and Bob both compute their CF_A and CF_B and exchange them. Each participant then passes its own elements through the filter of the peer to learn its local elements. Once Alice learns $A \setminus B$, she sends the missing elements to Bob, and vice versa.

Similar to the other ASM-based algorithms, cuckoo filter-based reconciliation may falsely detect some common elements as local to the sending peer. Thus, the elements that are already known to the receiving peer may be unnecessarily sent to it. However, Luo et al. [21] prove that if no insertion failure happens on Alice's and Bob's sides, cuckoo filter-based set reconciliation does not send redundant elements to the receiving end. On the other hand, even when all the insertions succeed, cuckoo filters still have certain amount of false positives of lookup. This is important because the false positive rate in cuckoo filters causes false negatives of reconciliation. In other words, Alice may look up element $x \in A \setminus B$ in CF_B and hit a false positive. In that case, Alice falsely concludes that $x \in A \cap B$ and does not send x to Bob, which negatively affects the reconciliation accuracy. Luo et al. [21] derive the probability that a cuckoo filter-based set reconciliation has no false negatives as approximately

$$\left(1 - \left(\frac{|B|}{m_B \cdot 2^{f-1}}\right)\right)^{|A \setminus B|} \left(1 - \left(\frac{|A|}{m_A \cdot 2^{f-1}}\right)\right)^{|B \setminus A|} \quad (2)$$

Where m_A and m_B are the number of buckets in CF_A and CF_B, while f is the common size of cuckoo filter fingerprints in bits. As opposed to the other similar ASM-based protocols, in the case of cuckoo filter-based set reconciliation, Alice and Bob do not need to agree on the size of their cuckoo filters upfront. Instead, they build cuckoo filters proportional to the size of their sets (S_A and S_B) and irrespective to the difference set size (d). However, the parties must agree on the hash functions they use to determine candidate buckets.

E. Counting Cuckoo Filter (CCF)

Li et al. [18] propose counting cuckoo filter (CCF) as an extension of cuckoo filters originally proposed by Fan et al [36]. They extend the cuckoo filter data structure in so that they attach a fixed-size counter to each slot of the cuckoo filter. The counter is utilized when a cuckoo filter represents a multiset. Besides the fingerprint of a multiset element, each slot now also keeps the multiplicity of the element. The CCF-based set reconciliation protocol is similar to the CF-based one in so that Alice and Bob build their CCF_A and CCF_B, and exchange them. As opposed to set reconciliation with CF's, multiset reconciliation with CCF's uses a more sophisticated technique of inferring the mutual differences. In the case of multiset, Alice does not only need to learn what are her local elements, but she also needs to learn the possible multiplicity differences among the common elements. Li et al. [18] propose two techniques for Alice to learn the information she needs. In the first technique, Alice would pass her local set A through CCF_B that she previously receive to learn $A \setminus B$. In the process of querying CCF_B, she also checks the multiplicity field of the fingerprints that she hits, and multiplies her local elements accordingly. In the second technique, Alice uses previously built CCF_A and linearly scans its slots. For the non-empty ones, she flags the slot as either containing a fingerprint of a local element, or as containing a fingerprint of a common element that she needs to multiply. In the latter case, she also stores the multiplicity difference. Once when Alice knows its local elements and the multiplicity differences the rest of the reconciliation protocol follows the steps of the CF-based set reconciliation.

F. Error Correcting Codes-based Algorithms

Minsky et al. [11] proposed a family of set reconciliation algorithms that use univariate polynomials to represent sets in a manner analogous to Reed-Solomon error correcting codes [40]. In the core approach, Characteristic-Polynomial Interpolation (CPI), Alice and Bob maintain the characteristic polynomials $\mathcal{X}_A(Z)$ and $\mathcal{X}_B(Z)$ such that zeros of the polynomials represent the elements of their sets. The main observation is

$$\frac{\mathcal{X}_A(Z)}{\mathcal{X}_B(Z)} = \frac{\mathcal{X}_{A\cap B}(Z)\cdot\mathcal{X}_{A\setminus B}(Z)}{\mathcal{X}_{A\cap B}(Z)\cdot\mathcal{X}_{B\setminus A}(Z)} = \frac{\mathcal{X}_{A\setminus B}(Z)}{\mathcal{X}_{B\setminus A}(Z)} \qquad (3)$$

On the basis of this observation, they define the following set reconciliation protocol. First, Alice and Bob exchange their characteristic polynomials. Then, each party computes the ratio between the two characteristic polynomials and factors the result to learn the elements unique to the other party. After this step, both Alice and Bob know $A \cup B$. A downside of this defined protocol would be that Alice and Bob need to communicate their polynomials in their full over the network. Since their characteristic polynomials grow linearly with the sizes of sets, such a protocol could not benefit from scenarios where $d \ll |S_A| + |S_B|$.

Fortunately, Alice and Bob do not have to compute the ratio between their characteristic polynomials directly. Instead, they can evaluate their characteristic polynomials at a number of random (non-data) points and exchange these evaluations. The evaluations can then be divided by each other and used to interpolate the desired rational function. Importantly, Minsky et al. [11] show that the number of evaluations needed to recover d symmetric differences is exactly d. Hence, the overall communication needed for Alice to learn $B \setminus A$ is only $(b+1)d+b$ bits, where b is the size of the single set element encoding. Note that this makes CPI-based set reconciliation almost optimal in communication. Although the computational complexity of such defined CPI-based set reconciliation is $O(d^3)$, it can be significantly improved using the technique we describe in Section IV-B.

Dodis et al. [24] offer a somewhat analogous set reconciliation protocol that, instead of encoding the sets into their characteristic polynomials as in CPI-based methods, uses a Bose-Chaudhuri-Hocquenghem (BCH) codes-based construction called *PinSketch*. With some optimization, their approach allows for an $O(d^2)$ decoding algorithm.

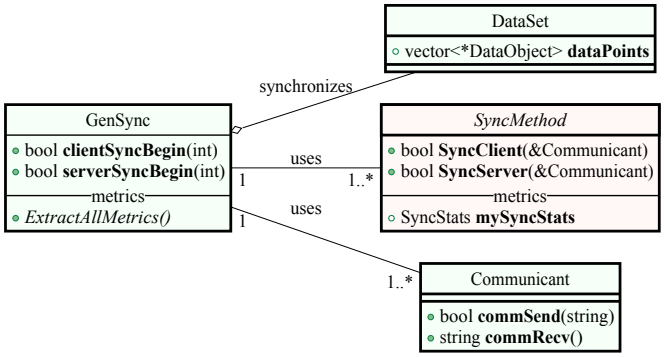


Fig. 1: Simplified UML diagram of the major components of the *GenSync* framework.

III. *GenSync* Framework

GenSync is a self-contained, open-source data reconciliation framework that is extensible by design and includes an integrated testbed. For the sake of portability, it is written in standard C++ and has two modes of operation. First, it can be compiled as a shared library and used as a building block of larger systems (*i.e., in-situ* mode). Second, it can be compiled with its integrated testbed and used for experimental evaluation (see Section III-B). Its current incarnation includes implementations of IBLT, cuckoo filter, and a variety of reconciliation protocols based on characteristic polynomial interpolation, including the core approach (CPI), interactive CPI (I-CPI), and prioritized CPI (P-CPI). As the current *GenSync* incarnation targets a wide range of devices, our algorithm implementations do not rely on platform-specific optimizations (*e.g.,* special CPU instructions and cache hierarchies). This design decision allows one to utilize *GenSync* in environments without unified platform requirements (*e.g.,* public blockchains). In Section V, we utilize the *GenSync* framework to evaluate the efficiency of the various protocols in a variety of practical settings.

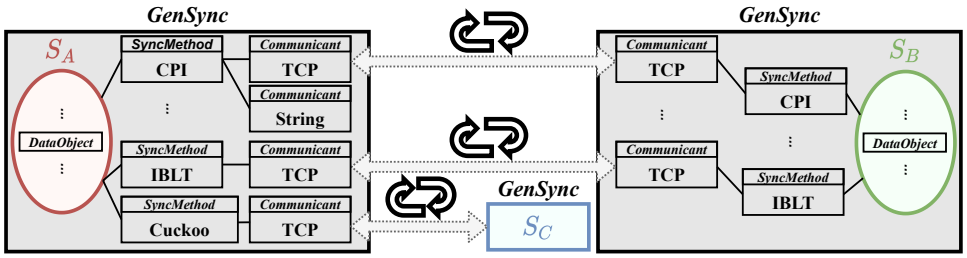


Fig. 2: *GenSync* usage example. Alice synchronizes her set S_A with both Bob (S_B) and Chuck (S_C). Each time Alice needs to synchronize with Bob, she can choose between CPI-over-TCP and IBLT-over-TCP. Alice uses Cuckoo-over-TCP for synchronization with Chuck.

A. Core Abstractions

GenSync's strength lies in its versatile structure, and we thus next discuss the purpose of the four core abstractions of the framework. The high-level relationship between these core abstractions is depicted in Fig. 1.

DataObject: The light-weight abstraction of a data point is called `DataObject`. *GenSync* requires each data point involved in reconciliation to be converted to a `DataObject`. A typical mapping between data points and `DataObjects` is implemented through data point hashing, where the resulting hashes consist the unique identifiers of `DataObjects`. With good hashing, we ensure the uniform distribution of the `DataObject` identifiers. However, for certain applications, such as *mempool* reconciliation (see Section VI), we may assume that the data (in that case Bitcoin transaction hashes) are already uniformly distributed binary strings that do not need an extra hashing layer. To support arbitrary length unique identifiers, we rely on Shoup's *Number Theory Library* [41] (NTL), a popular library that, among other data structures, supports variable length signed integers.

Communicant: The logic that allows *GenSync* to work with various underlying network protocols is encapsulated in the `Communicant` abstraction. The current version of the *GenSync* framework includes two `Communicants`: 1) TCP socket (used in this work), and 2) C++ string (for data sets that are disconnected in space, time, or reside on the same node). *GenSync* can easily be extended to support other network protocols (*e.g.,* UDP or SCTP — Stream Control Transmission Protocol [42]).

SyncMethod: The minimal common substructure for all data reconciliation protocols. It is designed to achieve three main goals:

1) agree on reconciliation parameters between hosts,
2) invoke data synchronization, and
3) update the host's underlying data set.

All the reconciliation protocols currently implemented in *GenSync* expose the same interface by inheriting from the `SyncMethod`, and the same can be done for future new data reconciliation protocols. The *GenSync* testbed (see Section III-B) relies on the `SyncMethod` abstraction to report performance metrics, that are used upstream in the framework. This allows for a standardized experimental comparison of existing and future protocols under different network and compute conditions.

GenSync: The topmost abstraction that connects the data sets, data reconciliation protocols, and communicants. When the *GenSync* framework is used *in-situ*, the `GenSync` object serves as the entry point. Typically, a user would include the *GenSync* library and create a `GenSync` object. The reconciliation starts when the peer that has invoked `SyncServer` receives a connection request from a peer that has invoked `SyncClient` [28].

Usage Example: As depicted in Fig. 1, the `GenSync` object may sequentially initiate data reconciliation with multiple peers, where each peer may use its own `Communicant` and `SyncMethod`. This enables simultaneous reconciliation of many hosts, possibly using different algorithms for different host-pairs. For example, when Alice needs to reconcile her data set with both Bob and Chuck, she may first probe the bandwidth on both Alice-Bob and Alice-Chuck connections. Based on her observations, Alice may decide to run one algorithm on the Alice-Bob and the other on Alice-Chuck connection. Fig. 2 depicts such a complex usage scenario.

B. GenSync *Testbed*

To evaluate the performance of the *GenSync* algorithms, we need a testbed that allows for 1) network parameters adjustments, and 2) isolation of the server and client execution environments. We address these requirements by combining our testbed with the Mininet [43] network emulator.

We use Mininet to emulate the network conditions (*i.e.,* bandwidth, latency and packet loss) between the two peers relying on the single-switch-two-nodes topology. For the purposes of the asymmetrical bandwidth emulation, we rely on the Traffic Control [44] feature of the Linux kernel.

Additionally, Mininet exposes access to core Linux kernel virtualization functions, which allow for execution environment isolation. For the purpose of the *GenSync* testbed, we rely on Linux `cgroup` capabilities and the kernel's Completely Fair Scheduler [45]. In conjunction with the information from the *PassMark* benchmark database [46], we constrain the computing power of the nodes to reflect the single-core performance of the modeled CPU's. More precisely, our testbed configures the `cgroup` parameters to assure that the emulated nodes consume only the desired fraction of the *host machine's* CPU single core capacity.

Limitations: The host machine and the virtualization techniques that we use impose some technical limitations to

Algorithm class	Communication	Computation	Multiset	Set of sets	Prioritized	Interactive
CBF [26]	$O(d)$	$O(\|S_A\| + \|S_B\|)$	✓	✗	✗	✗
CPI [11]	$(b+1)d + b$	$O(d^3)$	✗	✗	✓	✓
BCH [24]	$d \cdot b$	$O(d^2)$	✗	✗	✗	✗
IBLT [19]	$O(d)$	$O(d)$	✓ [35]	✓ [49]	✗	✓
CF [21]	$O(\|S_A\| + \|S_B\|)$	$O(\|S_A\| + \|S_B\|)$	✓(via CCF [18])	✗	✗	✗

TABLE II: Asymptotic communication and computation complexities for the set reconciliation algorithms from the literature. For each algorithm, we indicate its suitability for the four common variations of the set reconciliation problem.

the *GenSync* testbed. For example, the single core performance of the CPU that runs our testbed (*host machine*) imposes an upper limit on the compute power of the nodes we can emulate. In this work, we use a dedicated testbed host with an `Intel Core i7-7700` CPU, `v5.10.11` Linux kernel, and `v2.14.1` Open vSwitch [47]. On the other hand, the network parameters of the *GenSync* testbed are constrained by the capabilities of the Mininet emulator. For example, Mininet's packet loss model does not capture network mobility.

Since our testbed is integrated into the *GenSync* framework, it allows users to independently experiment with the implemented algorithms. For instance, *GenSync* users can run the testbed within their own environments and adjust both computational and network parameters to gain insight beyond the configurations used in this paper. Moreover, the modularity of *GenSync* allows users to overcome technical limitations of the existing testbed. For instance, users may decide to execute the *GenSync* `Runners` [28] in a network emulator that better captures wireless network dynamics (*e.g.,* Mininet-WiFi [48]).

IV. Analytical Comparison

In this section we focus on various factors that affect the performance of set reconciliation algorithms. To this end, we evaluate the asymptotic communication and computation behavior of the algorithms included in *GenSync* and summarize these behaviors in Table II. We categorize the algorithms according to the four common variations of the set reconciliation problem;

1) *multiset* reconciliation (storing multisets not sets),
2) *sets of sets* reconciliation (storing sets of sets),
3) *prioritized* reconciliation (tagging elements with priorities), and
4) *interactive* reconciliation (reconciling in multiple rounds of communication)

For sake of brevity, we do not explicitly describe the reconciliation of multisets and sets of sets in this work.

We next focus on techniques that aid with symmetric difference estimation, followed by a discussion of the accuracy-performance tradeoffs in the IBF and IBLT-based algorithms.

A. Symmetric Difference Upper Bound

The problem of establishing a tight bounds on the number of mutual differences (d) between remote sets is common to many set reconciliation protocols including the ones based on IBF, CBF, IBLT, and non-interactive CPI-based protocols. Such a bound is significant because better initial upper bound estimates may reduce the amount of information transferred over the network during reconciliation. On the other hand, an underestimate of d may cause some reconciliation protocols to fail.

In the general case, the problem of determining the exact size of mutual differences between remote hosts requires as much communication as a full set reconciliation [50]. For that reason, it is practical to rely on the heuristic approaches such as comparing random samples [51] and min-wise sketches [52], [53]. Although these heuristics may suffice when there are many differences among the sets, their accuracy deteriorates as the ratio of the mutual differences to overall set size decreases.

To address this deficiency, Eppstein et al. [19] propose the strata estimator (an approach that resembles the probabilistic distinct elements counter of Flajolet and Martin [54]). In the context of b-bit words, the strata estimator E represents a hierarchy of b IBF's of some constant size C (say, $C = 80$ buckets). As the first step in the reconciliation protocol, Alice inserts her elements into her strata estimator such that each element x is inserted into $E_A[i]$, where i is the number of trailing zeros in the binary representation of x. Alice then sends her strata estimator E_A alongside with her main IBF to Bob. By the same token, Bob builds his E_B and then computes $E_B[i] - E_A[i]$ for $i \in \{b-1, ..., 0\}$.

At each step, Bob decodes $E_B[i] - E_A[i]$. If decoding succeeds, he adds the count of the decoded elements to a global sum c. Otherwise, Bob terminates the process and estimates the number of differences as $2^{i^*}c$, where i^* is the level at which the decoding process failed. After Bob has estimated d, he can build his own main IBF and continue the reconciliation as in Section II-C.

The strata estimator gives accurate estimations of d only when d is small relative to the set sizes. In other cases, one may opt for a carefully gauged combination of strata estimator and Min-wise sketches that sums up to a predetermined constant size [19], [55]. Importantly, not all the reconciliation protocols included in *GenSync* need an initial set difference size estimate. Interactive CPI that we discuss next is one such example.

B. Interactive CPI

Minsky and Trachtenberg [27] have proposed a variation of the base CPI approach called partitioned set reconciliation. This is a "divide-and-conquer" algorithm that relies on recursive partitioning of the space of all bitvectors of size b

until a CPI reconciliation with some constant upper bound on symmetric differences size ($\overline{m}$) succeeds.

The "divide-and-conquer" algorithm begins by trying to reconcile all the differences at once using a small fixed $\overline{m}$. As this may fail due to an underestimated $\overline{m} < d$, Alice and Bob partition the bitvector space into p non-overlapping sub-partitions, $P_i,\ i \in \{1..p\}$, and invoke CPI on each of the sub-partitions. During the process of recursive partitioning, both Alice and Bob maintain an auxiliary data structure called *p-tree*. The nodes in p-tree represent the sub-partitions and each sub-partition can be in one of the three states; 1) *active* (sub-partition is currently being reconciled), 2) *terminal* (reconciliation succeeded at this sub-partition), and 3) *inactive* (reconciliation not yet attempted at this sub-partition). The algorithm completes when the entire bitvector space gets covered by terminal sub-partitions. The correctness of this approach relies on the fact that for all pairs of sets (S_A, S_B) and their non-overlapping partitions P_i such that $S_{A,B} = \cup_i P_i$, where $i \in \{0..p\}$, we can reconcile each $S_A \cap P_i$ with $S_B \cap P_i$ individually. That is, $S_A \cap P_i = S_B \cap P_i$ for all i implies $S_A = S_B$. As the described protocol involves multiple rounds of communication, one per each active or terminal node in the p-tree, we refer to the partitioned reconciliation approach as interactive CPI (I-CPI).

It should be noted that not all the set reconciliation approaches implemented by *GenSync* have an embedded mechanism for detecting reconciliation failure. For example, the cuckoo filter-based protocol as described in Section II-D does not include a mechanism to detect that the reconciliation has failed. On the other hand, CPI-based algorithms can detect that the reconciliation has failed simply by observing that the rational function interpolation has failed due to an insufficient number of evaluation points [11].

I-CPI With Priorities: Jin et al. [20] propose P-CPI as a CPI-based set reconciliation protocol for the sets where some elements have higher priority than the others. The protocol guarantees that the elements with a higher priority get reconciled first and considers the scenario in which Alice and Bob communicate over a disruptive network (*i.e.,* the communication channel may be cut at any moment). In the moment of the network failure, when Bob becomes unreachable, Alice would prefer to have received the maximal fraction of the high-priority elements from $S_B \setminus S_A$. In other words, if the reconciliation protocol gets interrupted at some point before the reconciliation of all the high-priority elements has completed, all the reconciled elements will be the high-priority ones. Conversely, if any low-priority elements are reconciled, then *all* the high-priority elements are already reconciled before that.

P-CPI is also a "divide-and-conquer" approach and can be seen as a generalization of I-CPI of Minsky and Trachtenberg [27]. In P-CPI, Alice's and Bob's sets are first divided into subsets by priority of their elements and the ratio of the high-priority elements to the size of the set is denoted as η (when $\eta = 1$, P-CPI is equivalent to I-CPI). Next, the subsets are sorted in the decreasing order of their priorities and I-CPI is invoked on each subset as previously described in Section IV-B. Based on the results in [27], the maximum number of CPI invocations is

$$I(m) \leq 1 + \frac{m}{\overline{m}} p \lceil b \log_p(2) \rceil \tag{4}$$

Taking into account that one invocation of CPI requires computation that is $\Theta(b\overline{m}^3 + b\overline{m}k)$ [27], we have that the worst-case computational complexity of P-CPI is [27]

$$\Theta\left(m\overline{m}^2 b^2 \frac{p}{\log p}\right).$$

From the same reference, we see that the *worst-case* communication complexity of P-CPI is $\Theta(m\frac{p}{\log p}b^2)$, since the communication complexity of CPI is $\Theta(mb)$. However, for certain applications, such as the one from Section VI, we can safely assume that the set differences are uniformly distributed over $GF(2^b)$. In that regards, Jin et al. [20] give the following theorem.

Theorem 1: When there are two sets with ηm uniformly distributed symmetric differences, P-CPI reconciliation makes $O(\eta m \log(\eta m))$ invocations to CPI, with probability at least $1 - \frac{1}{\eta m}$.

As a corollary, the computational cost of I-CPI is $O(m \cdot \eta b\overline{m}(\overline{m}^2 + k) \cdot \log(\eta m))$ with high probability. To generalize and clarify the conclusions of Jin et al. [20], we give a revised proof of *Theorem 1* in Appendix A. We further demonstrate the practical benefits of I-CPI in Section V-E as the part of our experimental evaluation.

C. IBF and IBLT

To make IBF's and IBLT's practical for set reconciliation, one needs to ensure that their decoding ("peeling") operation succeeds with a high probability. Otherwise, Alice and Bob will not be able to fully list the elements from the result of the subtraction between their corresponding IBF's. A failure in listing the result of the subtraction results in the failure of the reconciliation protocol. To this end, we must set the appropriate parameters so that IBF_A and IBF_B are sufficiently large at the start of reconciliation. At the same time, there is a desire to minimize the communication cost of the reconciliation protocol, leading to a preference for smaller IBF's.

Eppstein and Goodrich [19] have shown that for a fixed d, the listing of $IBF_A - IBF_B$ fails with probability at most $O(d^{-k})$, where k is the number of hash functions used in IBF's. Since the number of cells in the IBF's is proportional to d, this implies that larger IBF's have a lower probability of decoding failure.

For certain applications, it is desirable to optimize over a fixed decode success rate (p). In such cases, one may fix d and p and exhaustively search through the space of IBF parameters to find the optimally small ones. For the difference set sizes d below 1000 and the minimum of 95% probability of decode success, Ozisik et al. [1] have inferred the parameters that yield optimally small IBF's using such an exhaustive search. We will use these optimized parameters in our experimental evaluation of Section V.

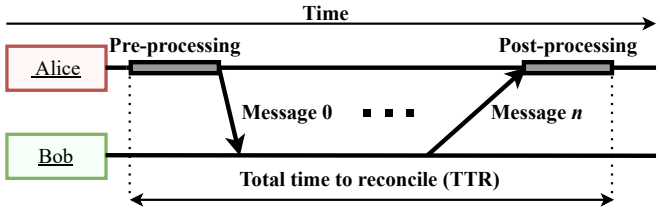


Fig. 3: Generic set reconciliation protocol structure and total time to reconcile (TTR).

Config. Name	Latency (ms)	Bandwidth (Mb/s)	Packet loss (%)
Mobile Broadband			
I	30	18	10^{-1}
II	20	35	10^{-3}
$III(a)$	**1**	20	10^{-1}
$III(b)$	**1**	**0.1**	10^{-1}
Consumer Internet			
IV	10	60	10^{-1}
$IV(a)$	10	**12/16 (uplink/downlink)**	10^{-3}

TABLE III: Network configurations chosen to mimic both mobile broadband and consumer internet networks. We consider both symmetrical and asymmetrical bandwidth internet connections.

V. Experimental Performance Comparison

We next conduct a series of experiments to assess the performance of the *GenSync* protocols under practical networking and computational constraints. In particular, we are interested in measuring:

1) *communication cost*;
2) *total time to reconcile* (TTR).

We define the *communication cost* as the number of bits transmitted between Alice and Bob until $S_A == S_B$, and TTR as the total wall-clock time elapsed until this happens (see Fig. 3). More precisely, we start our measurements at the moment that Alice initiates the process and stop when Alice learns that the set reconciliation has succeeded. Depending on the specific reconciliation protocol being used, Alice may learn this information on her own or through a status message received from Bob. However, for the purpose of this work, we set the parameters of all benchmarked algorithms such that the reconciliation succeeds for all experimental runs.

Config. Name	Alice's CPU	Bob's CPU
$\mathcal{B}$	Apple A13 Bionic @2.7 GHz	Apple A13 Bionic @2.7 GHz
$\mathcal{C}$	Intel Core i7-7700 @3.6 GHz	Intel Core i7-7700 @3.6 GHz
$\mathcal{CA}$	Intel Core i7-7700 @3.6 GHz	Samsung Exynos 9810 @2.3 GHz
$\mathcal{CB}$	Intel Core i7-7700 @3.6 GHz	Apple A13 Bionic @2.7 GHz

TABLE IV: Compute configurations include both symmetrical (*i.e.,* $\mathcal{A}$, and $\mathcal{B}$) and asymmetrical (*i.e.,* $\mathcal{CA}$ and $\mathcal{CB}$) compute scenarios.

The network configurations that we consider in this work are given in Table III. Configurations I and II roughly emulate mobile broadband networks with different performance and their parameters are derived from available measurement research [56], [57]. Configurations $III(a)$ and $III(b)$ correspond to low-latency-low-bandwidth networks that will be of interest later in Section V-D. Configurations IV and $IV(a)$ emulate typical internet connections and their parameters are drawn from popular low-end residential internet provider plans. Note that IV emulates a *symmetrical* bandwidth scenario whereas $IV(a)$ emulates an *asymmetrical* one that, for example, may occur in some DOCSIS [58] implementations.

Computational constraints: To model various computational constraints, we use the four compute configurations shown in Table IV. Note that we consider both *symmetrical* and *asymmetrical* computation. The former refers to the process of reconciliation between the devices with equally powerful CPU's, the latter refers to the scenarios where devices have CPUs of differing power. We use the single-letter configuration names to denote the symmetrical cases and two-letter names to denote the asymmetrical ones. We also use the ordered pair (N, C) to denote the configuration that consists of network configuration N and compute configuration C.

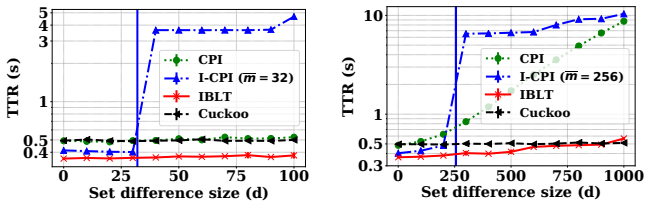
A. Impact Of Set Difference Size

In the first set of experiments, we fix the set sizes, compute, and network configurations for both Alice and Bob while varying the difference size (d). Each experiment is repeated 100 times and the measured quantities are reported with 95% confidence interval.

As shown in Fig. 4(a), IBLT requires the least amount of time to complete the reconciliation for the small numbers of set differences relative to the set sizes. Both CPI and cuckoo protocols perform around 40% worse than IBLT in these scenarios. As suggested by the results in Fig. 5(a), the TTR performance of the cuckoo protocol can be attributed to its non-optimal communication cost, proportional to the size of the sets of Alice and Bob (see Table II). The TTR performance of CPI can be attributed to its computational complexity disadvantage over IBLT (*i.e.,* roughly cubic in d for CPI and linear for IBLT).

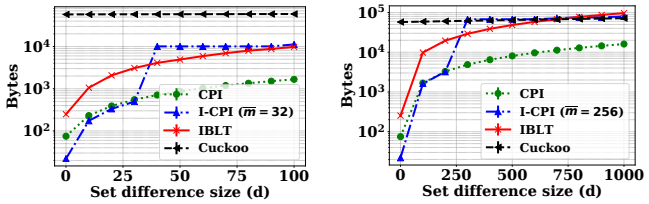
Note also that the TTR of I-CPI sharply increases at the point when it needs to perform multiple rounds of communication (*i.e.,* at $d = 32$, emphasized using vertical lines in Fig. 4). Intuitively, the more rounds of communication needed, the more time spent idle, waiting for data to arrive, relative to the calculation time of the CPI algorithm. In other words, when there are multiple rounds of communication, the network latency becomes the bottleneck.

Considering Fig. 4(b), we observe that, as the amount of mutual differences approaches 10% of the set sizes (*i.e.,* $d = 10^3$), the TTR performance of the cuckoo filter-based protocol approaches the TTR performance of the IBLT-based protocol. As suggested by Fig. 5(b), at the higher d values, the communication cost of IBLT and cuckoo approach each other.



(a) d ranges from 0 to 100. (b) d ranges from 0 to 10^3.

Fig. 4: TTR when $|S_A| = |S_B| = 10^4$ in the $(I, \mathcal{C})$ configuration (log scale).



(a) d ranges from 0 to 100. (b) d ranges from 0 to 10^3.

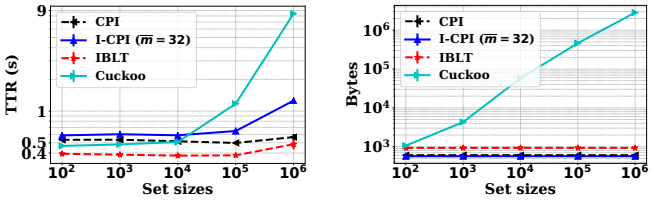
Fig. 5: Communication cost when $|S_A| = |S_B| = 10^4$ in the $(I, \mathcal{C})$ configuration (log scale).

Since IBLT and cuckoo-based protocols also have comparable computational complexities as d approaches $|S_A| = |S_B|$, this implies that when their communication costs converge so does their TTR performance.

Fig. 4(b) also suggests that when the mutual differences consist of a significant portion of the sets, the TTR performance of the CPI-based protocols deteriorates as the number of differences increases. That is, when the number of differences is sufficiently high, computation becomes the bottleneck for the CPI-based protocols. This effect conforms with our analysis in Section IV.

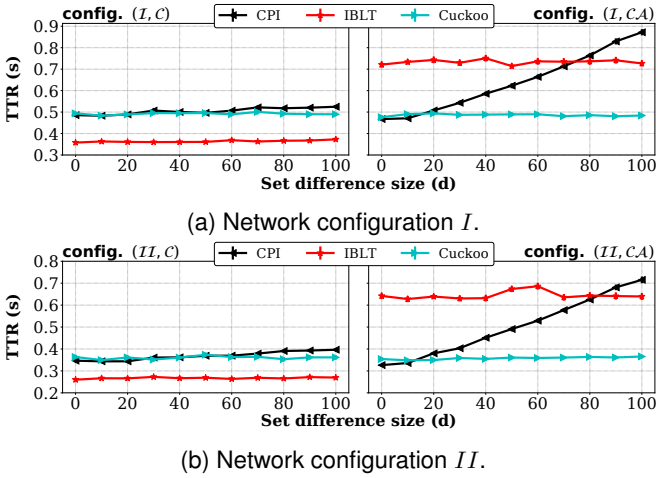
B. Impact Of Set Size

Next, we fix the number of differences (*e.g.,* $d = 30$) and vary the sizes of sets kept by Alice and Bob. For each set size, we run 100 experiments using the $(I, \mathcal{C})$ configuration (as described in Tables IV and III). As shown in Fig. 6(a), CPI-based protocols have a 25-300% worse TTR performance than IBLT as set size ranges from 100 to 10^6. Although the



(a) TTR (b) Communication

Fig. 6: Difference size $d = 30$. Set sizes $|S_A| = |S_B|$ vary from 100 to 10^6 (in $(I, \mathcal{C})$ configuration). TTR and communication cost shown in log scale.



(a) Network configuration I.

(b) Network configuration II.

Fig. 7: The impact of the network performance and available computing power to TTR-performance. Symmetric compute (left) versus asymmetric compute (right).

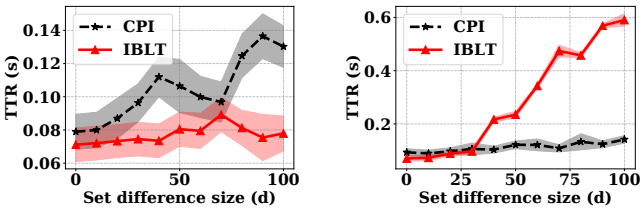
TTR performance of all the protocols deteriorates as the set sizes increase, IBLT remains dominant. Note that the TTR performance of the cuckoo-based protocol is comparable to the performance of CPI-based protocols as long as the set sizes are sufficiently small. Once when the sets reach a critical size, the TTR performance of cuckoo drastically deteriorates as the size of the sets increases. On the other hand, the TTR performance of the CPI protocol only slightly worsens for large sets.

C. Impact Of Compute And Network Parameters

To compare the *GenSync* protocols across various network and compute contexts, we fix the size of reconciling sets to 10^4 and consider relatively small numbers of mutual differences (*i.e.,* from 0 to 100). For such defined sets, we emulate the reconciliation between the following devices: 1) server-to-server ($\mathcal{C}$), 2) low-end smartphone and the server ($\mathcal{CA}$), and 3) high-end smartphone and the server ($\mathcal{CB}$) (see Appendix B). Additionally, each of these compute configurations is emulated in the context of networks I, II, IV, and $IV(a)$ from Table III. Further in this section, we outline the major network and compute parameters that affect the TTR performance of the *GenSync* algorithms.

Comparing the two system configurations (**config.** $(I, \mathcal{C})$ and **config.** $(I, \mathcal{CA})$) from Fig. 7(a) to their counterparts (**config.** $(II, \mathcal{C})$ and **config.** $(II, \mathcal{CA})$) from Fig. 7(b), we observe that the network performance has a significant impact on the performance of the reconciliation protocols. All the protocols that we consider improve their average performance under better network conditions (*i.e.,* network configuration II). The improvement ranges from 16% (for IBLT in compute configuration $\mathcal{CA}$) to 43% (for Cuckoo in compute configuration $\mathcal{C}$).

Even more important is that there is no dominant reconciliation protocol across all the compute configurations. While for the compute configuration $\mathcal{C}$ (symmetric compute), IBLT-based protocol exhibits the best TTR performance, for the compute configuration $\mathcal{CA}$ (asymmetric compute), cuckoo-



(a) 20 Mb/s bandwidth $III(a)$ (b) 0.1 Mb/s bandwidth $III(b)$

Fig. 8: TTR for $|S_A| = |S_B| = 10^4$. Compute configuration $\mathcal{C}$. 95% confidence interval. 100 iterations per point.

based protocol is dominant. As we can see in **config.** $(II, \mathcal{CA})$) from Fig. 7(b), the cuckoo filter-based protocol overtakes IBLT for all set difference sizes and both network configurations (*i.e.,* I and II). For $20 < d \leq 100$, cuckoo also dominates CPI, while for very small set difference sizes (*i.e.,* $d < 20$), cuckoo and CPI have comparable performance.

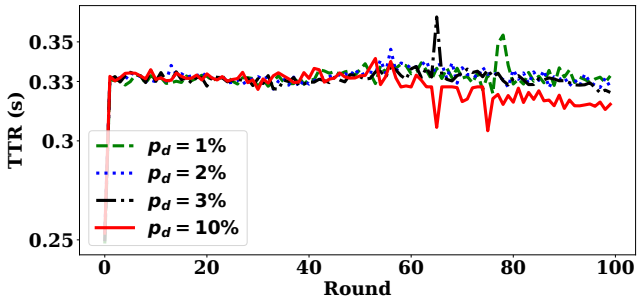
The domination of the cuckoo-based protocol in the conditions of *highly* asymmetrical compute (*i.e.,* $\mathcal{CA}$) appear to be related the following factors; 1) faster decode time of cuckoo filter relative to IBLT, and 2) relatively small size of sets with respect to the available bandwidth. As previously mentioned in Sections II-E, the decoding process of cuckoo filters consists of looking up each set element, where each lookup can be done in constant time. However, the decoding process of IBLT requires a computationally more intensive process often referred to as 'peeling". Since the TTR performance is bounded by the computation power of the computationally less efficient peer, the simplicity of cuckoo lookup procedure results in a better TTR performance. On the other hand, the inefficient communication of the cuckoo-based protocol is not a bottleneck as long as the network bandwidth is sufficiently large relative to the set sizes.

Unlike the case of asymmetrical compute power of the nodes, asymmetrical network bandwidth is not a decisive factor in determining the best *GenSync* reconciliation protocol as long as the bandwidth in any direction is not a bottleneck. The experimental results that support these conclusions are presented in Appendix B. In the next section, we discuss the best protocol choice when there is a bandwidth bottleneck.

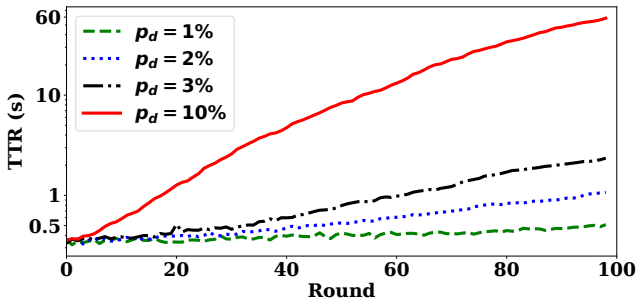
D. Impact Of Low Bandwidth

Besides the asymmetrical compute, the best reconciliation protocol choice may also be determined by the network conditions. For example, when a low-latency network offers just a fraction of its bandwidth to the reconciliation protocol. To capture such a scenario, we construct the two low latency network configurations $III(a)$ and $III(b)$ as in Table III, and run our experiments in the compute configuration $\mathcal{C}$, while keeping the set sizes at 10^4.

As depicted in Fig. 8(a), when the full bandwidth is dedicated to the reconciliation protocol, IBLT performs no worse than CPI for all values of d. However, when the available bandwidth drops to $0.1Mb/s$, as in Fig. 8(b), the performance



(a) I-CPI



(b) CPI (log scale)

Fig. 9: TTR in each round. Configuration $(II, \mathcal{C})$.

of IBLT deteriorates significantly. Although for $0 < d \leq 30$ the TTR of CPI and IBLT are comparable, for $d > 30$ CPI dominates IBLT and at $d = 100$ becomes 5x better than IBLT. The linear increase of TTR in the case of IBLT-based protocol may be attributed to its sub-optimal communication cost. On the other hand, as shown in [11], the CPI-based protocol has near optimal communication cost. When bandwidth is the bottleneck, the communication-optimal protocol also becomes the best protocol choice.

E. Benefits Of Interactive Protocols

The high-level applications that utilize *GenSync* may use its set reconciliation protocols in two distinct ways; 1) *cold start*, and 2) *incremental* reconciliation. The first case occurs when an application chooses not to keep any reconciliation-related data between two invocations to the reconciliation protocol (*e.g.,* to optimize for memory usage). The second case occurs when an application reconciles the same data set repeatedly many times and chooses to keep the reconciliation-related state between the invocations to the reconciliation protocol. In this section, we produce a set of experiments for comparing the effectiveness of interactive and non-interactive protocols for incremental reconciliation. In particular, we will focus on CPI and its interactive counterpart I-CPI.

As a matter of background, recall that interactive protocols such as I-CPI maintain reconciliation state over time (as described in Section IV-B). In the case of I-CPI, the state is represented as a p-ary tree that evolves as the reconciling sets change. The first invocation to I-CPI creates the p-ary tree, while each subsequent invocation operates on an existing data

structure and thus spends less time in the pre-processing phase. We proceed to demonstrate the comparative advantage that I-CPI has over CPI in the context of incremental reconciliation.

Consider a reconciliation between Alice and Bob where they agree to reconcile after they each add $N = 100$ elements to their sets, and stop when the sets reach 10^4 elements (*i.e.,* there will be 100 rounds of reconciliation in total). We make a few simplifying assumptions to aid with our evaluation: (i) we assume that additions to S_A and S_B happen in parallel at roughly the same time, and (ii) each element x that is being added to S_A is either new to Alice (*i.e.,* $x \notin S_A$), or new to both Alice and Bob (*i.e.,* $x \notin S_A \cup S_B$). This is likewise the case for any y being added to S_B. We define p_d to be the probability that any elements (x, y), where x is about to be added to S_A and y to S_B, are new both to Alice and Bob (*i.e.,* $\{x, y\} \not\subset S_A \cup S_B$). Note that such x and y will end up as the symmetric differences between Alice and Bob.

For various values of p_d, we evaluate CPI and I-CPI protocols while keeping our testbed in configuration $(II, \mathcal{C})$ for all experiments. Fig. 9(a) shows that the TTR performance of I-CPI is close to constant across all rounds and almost identical for all values of p_d. That is, I-CPI exhibits a stable TTR performance as the set sizes increase as long as the number of differences being reconciled in each round is close to constant.

On the other hand, the TTR performance of the non-interactive version (CPI) is significantly worse than I-CPI when it comes to incremental reconciliation. As shown in Fig. 9(b), the TTR performance of CPI is negatively affected both by the size of the sets and by the higher values of p_d. These trends appear to be due to the fact that CPI does not keep any reconciliation-related state between the rounds. Thus, it cannot benefit from any differences already reconciled in the previous rounds. On top of that, there is another significant disadvantage of CPI over I-CPI in the context of incremental reconciliation; it requires an upper bound on the number of symmetric differences (d). Although in our experiments this information can be derived from p_d, in some practical applications p_d may not be known. As previously mentioned in Section IV-B, I-CPI does not require such a bound.

Besides I-CPI, *GenSync* offers other protocols that support interactive reconciliation. For example, the IBLT protocol may also be used in the interactive mode. When that is the case, then both Alice and Bob maintain their corresponding tables over time, inserting new elements as they arrive. However, as discussed in Section IV-C, the number of inserted elements may affect the IBLTs ability to decode, consequently affecting the accuracy of reconciliation. For this reason, it is necessary to ensure that IBLT's have sufficient capacity. One way to address this shortcoming is to periodically rebuild the IBLT as it fills up. As IBLTs are not adjustable in size, it is necessary recreate them by decoding the old table and reinserting its elements into the new table. The accuracy of I-CPI, on the other hand, does not deteriorate as the new elements arrive.

F. Summary of Findings

We summarize our overall insight into choosing the best protocol in *GenSync* as follows:

(A) IBLT is the best choice for symmetrical computational resources, sufficient bandwidth (at least tens of Mb/s), and a small ratio of set differences to set size ($\leq 10\%$).
(B) Cuckoo is the best choice for asymmetrical compute and set sizes close to 10^4 as long as bandwidth is sufficiently high (approximately tens of Mb/s).
(C) CPI is the best choice for the low latency (tens of ms) but low bandwidth (hundreds of Kb/s) conditions.

Among the protocols considered, CPI is the closest to the information-theoretic communication minimum [11].

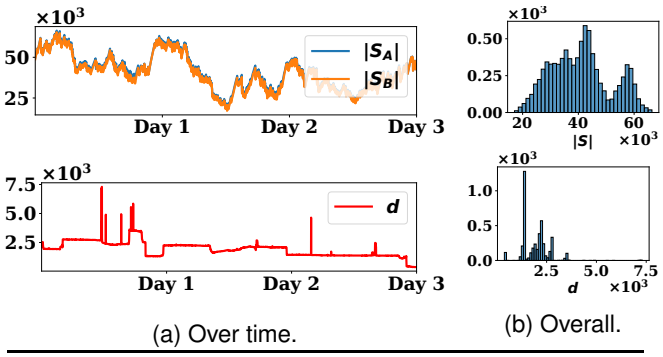
VI. Bitcoin Data Set Evaluation

We next describe the use of our framework on a practical data set taken from the Bitcoin network. We expect that this approach can be used as a template for evaluating reconciliation performance in other applications of interest.

Application Scenario : In the following experiments, we reconcile the *mempool*s of two neighboring Bitcoin full nodes. These pools maintain the Bitcoin transactions that a node has received but not yet included in a block of the underlying blockchain. More precisely, we conduct experiments that reconcile the sets of 32-byte long transaction identifiers generated by the two invocations of the `SHA256` hash function (*i.e.,* identifiers are uniformly distributed in $GF(2^{256})$). Our experimental setting is motivated by the results of Imtiaz et al. [3], which have recently shown that synchronization of *mempools* can significantly impact the average transaction propagation delay in the event of churn (intermittent connection between peers). To increase the synchronization among neighboring peers, they proposed a proof-of-concept scheme called *MempoolSync* and implemented it in the Bitcoin software. However, the authors did not attempt to make their scheme efficient. Instead, they suggested replacing their scheme with a set reconciliation protocol in the future. As *GenSync* supports comparison of various protocols, we utilize this framework to gain insight into which protocol would be a good replacement for *MempoolSync*. Note that *GenSync* could be integrated as a library in the Bitcoin software, if desired (as discussed in Section III).

Data Collection : To collect the experimental data sets, we have conducted a three day long measurement campaign on two live Bitcoin nodes and collected the snapshots of their *mempools* each minute. Altogether, we collected a data set that consists of 4320 pairs of sets (S_A and S_B) where each set represents the collection of non-verified Bitcoin transaction identifiers. Fig. 10 summarizes the outcomes of our measurement campaign. Note that the mean set size ($|S|$) is 40835, while the mean size of the set of symmetric differences (d) is 1930. On average, the two sets differ in about 4.7% of their elements. However, both the standard deviations of the set sizes and difference sizes are relatively high: 10654 and 656, respectively.

Results Analysis : To emulate our network of interest, we utilize the *GenSync* testbed in the configuration $(I, \mathcal{C})$ (see Section III-B for hardware/software specification) . In other words, we assume that the neighboring Bitcoin nodes have symmetrical computing power. We then sequentially execute

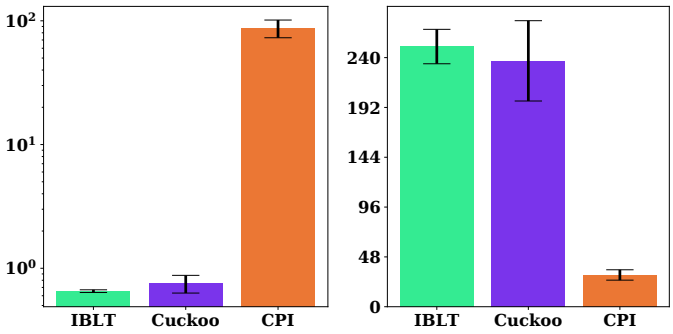


(a) Over time. (b) Overall.

	Set size ($\lvert S_A\rvert$ or $\lvert S_B\rvert$)	Difference size (d)	d / ($\lvert S_A\rvert$ or $\lvert S_B\rvert$)
min	17080	379	$\approx 0.76\%$
max	66401	7300	$\approx 16.7\%$

(c) Extreme values.

Fig. 10: Distribution of *mempool* sizes ($|S_A|$ and $|S_B|$) and symmetric difference sizes (d).



(a) TTR (s) (b) Communication cost (KB)

Fig. 11: *GenSync* protocols applied to *mempool* reconciliation. Configuration $(I, \mathcal{C})$. 95% confidence intervals.

the reconciliation protocol on each pair of sets in the cold start mode and measure the TTR and communication cost. As shown in Fig. 11(a), the IBLT protocol exhibits the best mean TTR performance among the benchmarked *GenSync* protocols (around $600ms$ with a low variance). Applied to the problem of *mempool* reconciliation, this result may suggest an initial estimate on the maximum frequency of reconciliation. Roughly speaking, since IBLT reconciles the *mempools* within a second, Bitcoin full nodes may decide to reconcile their *mempools* each second.

However, as shown in Fig. 11(b), the communication cost of IBLT is only slightly worse than cuckoo, but significantly inferior when compared to CPI. Thus, there is a trade-off between the maximum frequency of reconciliation and the consumed bandwidth. Additionally, as we discussed in Sections V-C and V-D, the latency and bandwidth variations of the network of interest should also be taken in consideration.

It is also important to emphasize that cuckoo protocol has a better mean communication cost when compared to IBLT for this data set. Considering only the analytical results from Table II, this may seem surprising. The communication cost of cuckoo filter-based reconciliation protocol is $O(|S_A| + |S_B|)$, while the communication cost of the IBLT-based protocol is only $O(d)$. However, as we discussed earlier in Section V-A, as the ratio between the difference size and the set sizes increases, the actual communication costs of the two protocols converge. Recalling Fig. 5(b), as this ratio exceeds 10%, the actual communication cost of IBLT becomes worse than cuckoo. Since for the data set in Fig. 10 and the $(I, \mathcal{C})$ network and computation parameters cuckoo filter-based protocol achieves a comparable mean TTR performance (under 1s), it may be a plausible alternative to IBLT. Furthermore, if we need to optimize for bandwidth, then CPI-based methods should also be considered. As we have demonstrated in Section V-E, the TTR performance of CPI-based methods can be significantly improved when its interactive version is utilized. However, this optimization comes at a cost of increasing the rounds of communication. When the network latency is high, multiple rounds of communication may significantly affect the TTR performance. This trade-off has been discussed in Section V-A.

VII. Conclusion

In this work, we produced a taxonomy of several prominent set reconciliation approaches and compared their analytical properties focusing on communication and computation complexity. For the purpose of a standardized experimental evaluation, we introduced a novel metric called *total time to reconcile* (TTR) that captures key system properties, such as network conditions and compute capabilities of the peers. To assist with experimental evaluation, we produced *GenSync*, an open-source data reconciliation framework with an integrated testbed that unifies a number of popular protocols. Unlike previous approaches, *GenSync* allows for a methodical comparison among the protocols in a wide range of practical system conditions. Thus, *GenSync* can be utilized for determining the best protocol for a given system model. Thanks to its modularity, *GenSync* can both serve as a benchmarking tool and as a library to be integrated in existing software. We have leveraged the *GenSync* testbed to evaluate each reconciliation protocol across a set of exemplar network and compute parameters.

In all, we found that the optimal choice of the reconciliation protocol depends significantly on a number of factors, including statistical properties of data being reconciled, network bandwidth and latency, symmetry of the computing power of peers, and whether reconciliation is done in *cold start* or *incremental* mode. Indeed, our experiments show that (i) IBLT-based reconciliation exhibits the best performance when bandwidth is not a bottleneck and peers have similar computational power; (ii) cuckoo-based reconciliation dominates when bandwidth is sufficiently high but there is a gap in computational power of peers, and (iii) CPI-based approaches perform best when there is a bandwidth bottleneck.

More generally, our experiments have shown that a poor choice of reconciliation protocols may lead to up to a *5x* hit in performance. Moreover, when it comes to incremental reconciliation, we have found that the interactive protocols (*e.g.,* I-CPI) may exhibit multiple orders of magnitude better performance than their non-interactive counterparts. We

showcased some of these effects (and the evaluative power of *GenSync*) concretely through a measurement campaign on two live Bitcoin full nodes, examining the effects of different reconciliation protocols for *mempool* reconciliation.

Our conclusions demonstrate that the reasoning about the performance of set reconciliation protocols in practical contexts is a complex task that can benefit from an integrated framework and testbed such as *GenSync*.

An important area for future work is to extend our framework to include adaptive set reconciliation protocols. Although some systems are designed for a narrow range of system parameters, other need to tolerate larger parameter fluctuations (*e.g.,* bandwidth and available compute) without interruption or losing performance. Future work should focus on meeting the requirements of such systems through adaptive solutions that can switch between reconciliation protocols dynamically in response to large fluctuations of system parameters.

ACKNOWLEDGMENT

We want to thank Sean Brandenburg, Eliezer Pearl, Zifan Wang, and Shubham Arora for contributing code to an earlier version of *GenSync*. Furthermore, we want to thank the anonymous reviewers for the valuable comments and suggestions that lead us to the final version of this work.

REFERENCES

[1] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to blockchain propagation," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 303–317, doi: https://doi.org/10.1145/3341302.3342082.

[2] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 817–831, doi: https://doi.org/10.1145/3319535.3354237.

[3] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the bitcoin network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1598–1615, 2021, doi: https://doi.org/10.1109/TNSM.2021.3050428.

[4] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, "Robust set reconciliation," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 135–146, doi: https://doi.org/10.1145/2588555.2610528.

[5] M. Mitzenmacher and R. Pagh, "Simple multi-party set reconciliation," *CoRR*, vol. abs/1311.2037, 2013. [Online]. Available: http://arxiv.org/abs/1311.2037

[6] M. Stonebraker, "In search of database consistency," *Commun. ACM*, vol. 53, no. 10, p. 8–9, oct 2010, doi: https://doi.org/10.1145/1831407.1831411.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205–220, oct 2007, doi: https://doi.org/10.1145/1323293.1294281.

[8] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 143–157, doi: https://doi.org/10.1145/2043556.2043571.

[9] S. J. Roy, D. Grenader, and O. Lvovitch, "Managing operations between heterogeneous file systems," Jan. 24 2019, US Patent App. 16/040,375.

[10] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "Sporc: Group collaboration using untrusted cloud resources," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, p. 337–350, doi: https://doi.org/10.5555/1924943.1924967.

[11] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," in *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No.01CH37252)*, 2001, pp. 232–, doi: https://doi.org/10.1109/ISIT.2001.936095.

[12] K. P. Puttaswamy, C. C. Marshall, V. Ramasubramanian, P. Stuedi, D. B. Terry, and T. Wobber, "Docx2go: Collaborative editing of fidelity reduced documents on mobile devices," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 345–356, doi: https://doi.org/10.1145/1814433.1814467.

[13] T. Chen, D. Guo, X. Liu, H. Chen, X. Luo, and J. Liu, "Bdp: A bloom filters based dissemination protocol in wireless sensor networks," in *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, 2009, pp. 593–602, doi: https://doi.org/10.1109/MOBHOC.2009.5336950.

[14] M. Karpovsky, L. Levitin, and A. Trachtenberg, "Data verification and reconciliation with generalized error-control codes," *IEEE Transactions on Information Theory*, vol. 49, no. 7, pp. 1788–1793, 2003, doi: https://doi.org/10.1109/TIT.2003.813498.

[15] A. Orlitsky, "Communication issues in distributed computing," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, 1986.

[16] D. Starobinski, A. Trachtenberg, and S. Agarwal, "Efficient pda synchronization," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, pp. 40–51, 2003, doi: https://doi.org/10.1109/TMC.2003.1195150.

[17] H. Yan, U. Irmak, and T. Suel, "Algorithms for low-latency remote file synchronization," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 156–160, doi: https://doi.org/10.1109/INFOCOM.2008.40.

[18] S. Li, L. Luo, D. Guo, and Y. Zhao, "Multiset synchronization with counting cuckoo filters," in *Wireless Algorithms, Systems, and Applications*, D. Yu, F. Dressler, and J. Yu, Eds. Cham: Springer International Publishing, 2020, pp. 231–243, ISBN: 978-3-030-59016-1.

[19] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? efficient set reconciliation without prior context," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 218–229, doi: https://doi.org/10.1145/2018436.2018462.

[20] J. Jin, W. Si, D. Starobinski, and A. Trachtenberg, "Prioritized data synchronization for disruption tolerant networks," in *MILCOM 2012 - 2012 IEEE Military Communications Conference*, 2012, pp. 1–8, doi: https://doi.org/10.1109/MILCOM.2012.6415678.

[21] L. Luo, D. Guo, O. Rottenstreich, R. T. Ma, and X. Luo, "Set reconciliation with cuckoo filters," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2465–2468, doi: https://doi.org/10.1145/3357384.3358065.

[22] L. Luo, D. Guo, J. Wu, O. Rottenstreich, Q. He, Y. Qin, and X. Luo, "Efficient multiset synchronization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, p. 1190–1205, apr 2017, doi: https://doi.org/10.1109/TNET.2016.2618006.

[23] L. Gong, Z. Liu, L. Liu, J. Xu, M. Ogihara, and T. Yang, "Space- and computationally-efficient set reconciliation via parity bitmap sketch (pbs)," *Proc. VLDB Endow.*, vol. 14, no. 4, p. 458–470, dec 2020, doi: https://doi.org/10.14778/3436905.3436906.

[24] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540, ISBN: 978-3-540-24676-3.

[25] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, p. 281–293, jun 2000, doi: https://doi.org/10.1109/90.851975.

[26] D. Guo and M. Li, "Set reconciliation via counting bloom filters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2367–2380, 2013, doi: https://doi.org/10.1109/TKDE.2012.215.

[27] Y. Minsky and A. Trachtenberg, "Practical set reconciliation," in *40th Annual Allerton Conference on Communication, Control, and*

Computing, vol. 248, 2002. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.7200&rep=rep1&type=pdf

[28] N. Boskov, A. Trachtenberg, D. Starobinski, and contributors. GenSync Framework. [Online]. Available: http://www.github.com/nislab/gensync

[29] M. Skjegstad and T. Maseng, "Low complexity set reconciliation using bloom filters," in *Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing*, ser. FOMC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 33–41, doi: https://doi.org/10.1145/1998476.1998483.

[30] X. Tian, D. Zhang, K. Xie, C. Hu, M. Wang, and J. Deng, "Exact set reconciliation based on bloom filters," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 3. IEEE, 2011, pp. 2001–2009.

[31] L. Luo, D. Guo, Y. Zhao, O. Rottenstreich, R. T. B. Ma, and X. Luo, "Mcfsyn: A multi-party set reconciliation protocol with the marked cuckoo filter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2705–2718, 2021, doi: https://doi.org/TPDS.2021.3074440.

[32] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422 — 426, Jul. 1970, doi: https://doi.org/10.1145/362686.362692.

[33] S. Z. Kiss, E. Hosszu, J. Tapolcai, L. Ronyai, and O. Rottenstreich, "Bloom filter with a false positive free zone," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1412–1420, doi: https://doi.org/10.1109/INFOCOM.2018.8486415.

[34] D. Eppstein and M. T. Goodrich, "Straggler identification in round-trip data streams via newton's identities and invertible bloom filters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 2, pp. 297–306, 2011, doi: https://doi.org/10.1109/TKDE.2010.132.

[35] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2011, pp. 792–799, doi: https://doi.org/10.1109/Allerton.2011.6120248.

[36] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 75–88, doi: https://doi.org/10.1145/2674005.2674994.

[37] L. Luo, D. Guo, O. Rottenstreich, R. T. B. Ma, X. Luo, and B. Ren, "A capacity-elastic cuckoo filter design for dynamic set representation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4860–4874, 2021, doi: https://doi.org/TNSM.2021.3099433.

[38] F. Zhang, H. Chen, H. Jin, and P. Reviriego, "The logarithmic dynamic cuckoo filter," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 948–959, doi: https://doi.org/ICDE51399.2021.00087.

[39] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004, doi: https://doi.org/10.1016/j.jalgor.2003.12.002.

[40] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. Elsevier, 1977, vol. 16, ISBN: 978-0-444-85010-2.

[41] Victor Shoup and others, "NTL: A Library for doing Number Theory." [Online]. Available: https://libntl.org/

[42] R. Stewart and C. Metz, "Sctp: new transport protocol for tcp/ip," *IEEE Internet Computing*, vol. 5, no. 6, pp. 64–69, doi: https://doi.org/10.1109/4236.968833.

[43] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 253–264, doi: https://doi.org/10.1145/2413176.2413206.

[44] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, vol. 213. sn, 2002. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.620.1459&rep=rep1&type=pdf

[45] Linux Kernel, "Completely Fair Scheduler." [Online]. Available: https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html

[46] PassMark Software, "CPU Benchmarks." [Online]. Available: https://www.cpubenchmark.net

[47] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130, ISBN: 978-1-931971-218.

[48] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 384–389, doi: https://doi.org/10.1109/CNSM.2015.7367387.

[49] M. Mitzenmacher and T. Morgan, "Reconciling graphs and sets of sets," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. SIGMOD/PODS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 33–47, doi: https://doi.org/10.1145/3196959.3196988.

[50] S. Agarwal and A. Trachtenberg, "Approximating the number of differences between remote sets," in *2006 IEEE Information Theory Workshop-ITW'06 Punta del Este*. IEEE, 2006, pp. 217–221, doi: https://doi.org/10.1109/ITW.2006.1633815.

[51] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 604–613, doi: https://doi.org/10.1145/276698.276876.

[52] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000, doi: https://doi.org/10.1006/jcss.1999.1690.

[53] A. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, 1997, pp. 21–29, doi: https://doi.org/10.1109/SEQUEN.1997.666900.

[54] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985, doi: https://doi.org/10.1016/0022-0000(85)90041-8.

[55] M. Gentili, "Set reconciliation and file synchronization using invertible bloom lookup tables," 2015, Bachelor's Thesis, Harvard University, Cambridge, MA.

[56] N. Becker, A. Rizk, and M. Fidler, "A measurement study on the application-level performance of lte," in *2014 IFIP Networking Conference*, 2014, pp. 1–9, doi: https://doi.org/10.1109/IFIPNetworking.2014.6857113.

[57] C. Midoglu, L. Wimmer, A. Lutu, O. Alay, and C. Griwodz, "Monroe-nettest: A configurable tool for dissecting speed measurements in mobile broadband networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 342–347, doi: https://doi.org/10.1109/INFCOMW.2018.8406836.

[58] CableLabs, "DOCSIS Technology Specifications." [Online]. Available: https://www.cablelabs.com/technologies/docsis-4-0-technology

[59] D. Dubhashi and A. Panconesi, *Concentration of Measure for the Analysis of Randomized Algorithms*, 1st ed. USA: Cambridge University Press, 2009, ISBN: 0521884276.

APPENDIX A
PROOF OF THEOREM 1:

Our proof is somewhat similar to that of randomized Quicksort in [59] with a modified partitioning process that requires separate analysis.

Proof: Without the lost of generality, we consider a binary partition tree ($p = 2$) and write $m' = m'_0 = \eta m$ to denote the overall number of the high-priority differences. We call a node in the partition tree *good* if each of its children contains at least one third of its differences. Otherwise, the node is *bad*. If we now consider a t nodes long root-to-leaf path that consists of only good nodes, we will see that the number of differences at level t is

$$m'_t \leq \left(\frac{2}{3}\right) m'_{t-1} \leq \left(\frac{2}{3}\right)^t m'_0 \tag{5}$$

Since we know that each good node must contain at least one difference, we can use Eq.(5) to derive the maximal number of good nodes in any path as

$$t \leq \frac{\log_2 m'}{\log_2\left(\frac{3}{2}\right)} < 2\log_2 m' \tag{6}$$

Next step in our proof is to show that the following holds for any root-to-leaf path P that may contain both bad and good nodes

$$Pr[|P| > 4\log_2 m'] < \frac{1}{m'^2} \tag{7}$$

Let X_i be a random variable that takes value 1 when the i-th node on path P is bad, and 0 otherwise. If we denote the number of differences at node i as s_i, then by the definition of good nodes we have

$$Pr[X_i = 1] = 1 - \frac{\sum_{j=s_{i-1}/3}^{2s_{i-1}/3} \binom{s_{i-1}}{j}}{2^{s_{i-1}}} \leq \frac{2}{3} \tag{8}$$

Now, let X be the random variable that denotes the number of bad nodes along P. Since all X_i are independent, we can use Eq. (8) and the union bound to derive

$$E[X] = \sum_{i}^{|P|} X_i \leq \frac{2}{3}|P| \tag{9}$$

Then, we assume $|P| \geq \frac{3}{2e}\log_2 m'$, and apply the Chernoff-Hoeffding bounds for some $q > 2eE[X]$ as follows [59]:

$$Pr[X > q] \leq 2^{-q} \leq 2^{-2\log m'} = \frac{1}{m'^2} \tag{10}$$

Since the length of any path is the sum of its good and bad nodes (*i.e.,* $|P| = X + t$ by definition), we can combine Eq.(6) and Eq.(10) to show that Eq. (7) holds for all paths P

$$Pr[X > q] = Pr[|P| > 4\log m'] \leq \frac{1}{m'^2} \tag{11}$$

Importantly, for $|P| < \frac{3}{2e}\log_2 m'$, we have that Eq. (7) is trivially satisfied as $P[|P| > 4\log_2 m']$ is zero.

Next, we see that our partition tree can have at most m' leafs, since each leaf must contain at least one difference. Hence, the maximal number of root-to-leaf paths is m'. We can apply the union bound to prove that all the root-to-leaf paths will be shorter than $4\log_2 m'$ nodes with probability $1 - \frac{1}{m'}$ as follows

$$\begin{aligned} Pr[\exists P, |P| > 4\log_2 m'] &\leq m' \cdot Pr[|P| > 4\log_2 m'] \\ &\leq \frac{1}{m'} \end{aligned} \tag{12}$$

Finally, P-CPI calls CPI at each node of the partition tree. Thus, the overall number of CPI invocations is $O(m'\log m')$. ■

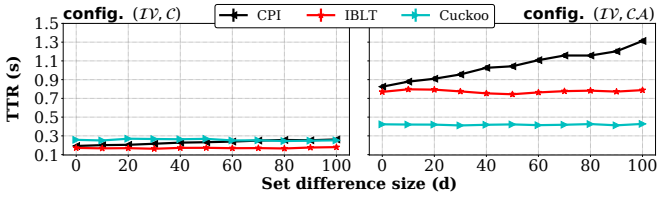
Note that in practice we usually have that $m \ll 2^b$, thus the worst-case number of CPI invocations of $O(mb)$ (see Eq. (4)) is worse than $O(m\log m)$ CPI invocations in the high-probability case from Theorem 1.

Combining the high-probability upper bound on the number of CPI invocations from Theorem 1 and the worst-case computation complexity of CPI from [27], we get that the computation complexity of P-CPI is $O(m{\cdot}\eta b\overline{m}(\overline{m}^2{+}k){\cdot}\log(\eta m))$, with probability $1 - \frac{1}{\eta m}$. By the same token, the communication complexity is $O(m{\cdot}\overline{m}\eta b{\cdot}\log(\eta m))$ with the same probability. Importantly, $\overline{m}$ is a fixed constant known in advance.

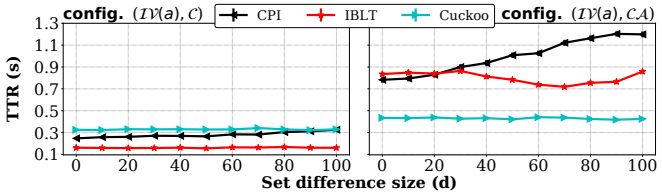
APPENDIX B
IMPACT OF ASYMMETRICAL NETWORK AND COMPUTE:

As indicated in Section V-C, asymmetry in the network bandwidth does not have a decisive effect on the best protocol choice as long as the bandwidth in any direction is not a bottleneck. In Fig. 12, we present experimental evidence that supports this claim. We run the same set of experiments as in Section V-C, but now we use the network configuration IV from Table IV (in Fig. 12(a)) and its asymmetrical bandwidth version $IV(a)$ (in Fig. 12(b)). Comparing **config.** $(IV, \mathcal{C})$ with **config.** $(IV(a), \mathcal{C})$, and **config.** $(IV, \mathcal{CA})$ with **config.** $(IV(a), \mathcal{CA})$, we observe that the order of the considered protocols remains the same. On the other hand, comparing the left parts of Fig. 12 to their counterparts on the right, we observe that asymmetrical *compute* may decide the best *GenSync* protocol (the effect we described in Section V-C).

In Fig. 13, we present the impact of network conditions on the *GenSync* protocols performance when reconciliation happens between a high-end smartphone and the server ($\mathcal{CB}$). First, comparing the left part of Fig. 13 to its right part, we observe that the better network conditions translate to a better TTR performance for all considered protocols (as we pointed out in Section V-C). However, comparing to **config.** $(I, \mathcal{C})$ and **config.** $(II, \mathcal{C})$ from Fig. 7, we observe that the best *GenSync* protocol does not change. That is, the IBLT-based protocol remains the best among the ones offered by the *GenSync* framework. The reason for this lies in the fact that our compute configurations $\mathcal{C}$ and $\mathcal{B}$ from Table IV have comparable single-thread performance according to the *PassMark* benchmarks [46]. According to these benchmarks, $\mathcal{B}$ has only a 17% weaker single-thread rating than $\mathcal{C}$. On the other hand, the compute power disrepancy between $\mathcal{A}$ and $\mathcal{C}$ is 3.5 times larger (*i.e.,* $\mathcal{A}$ has a 59% weaker single-thread rating than $\mathcal{C}$). As opposed to small compute power discrepancies (*e.g.,* $\mathcal{CB}$), the larger ones (*e.g.,* $\mathcal{CA}$) may have



(a) Network configuration IV — *symmetrical* bandwidth.



(b) Network configuration $IV(a)$ — *asymmetrical* bandwidth.

Fig. 12: The impact of asymmetrical *network bandwidth* to TTR-performance. Symmetrical *compute* (left) and asymmetrical *compute* (right).

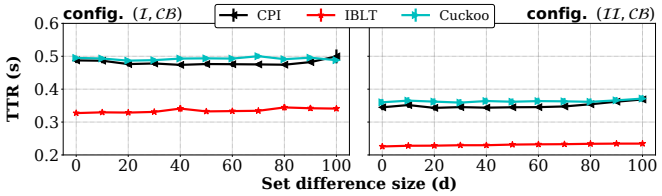


Fig. 13: The impact of the network conditions to TTR-performance for the $\mathcal{CB}$ compute configuration. Network configuration I (left) versus network configuration II (right). $\mathcal{CB}$ has a 3.5 times *smaller* compute power discrepancy than $\mathcal{CA}$.

a decisive impact on the best protocol choice (as we discussed in Section V-C).

Novak Boškov is a Ph.D. candidate in Computer Engineering at Boston University. His research interests include the performance optimizations of globally distributed systems and security in multi-tenant clouds. He recently focuses on improving information propagation in blockchains as well as the side-channel analysis of software container deployments in commercial cloud computing services.

Prior to joining Boston University in 2018, he has received his Bachelor with Honours and Master degrees in Electrical and Computer Engineering from University of Novi Sad, Serbia in 2015 and 2016, respectively.

Ari Trachtenberg received the S.B. degree from MIT in 1994, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign (UIUC) in 1996 and 2000, respectively. He is a Professor of Electrical and Computer Engineering, Computer Science, and Systems Engineering at Boston University (BU), where he has been since September 2000. He has also been a Distinguished Scientist Visitor with Ben Gurion University, a Visiting Professor with the Technion — Israel Institute of Technology, and worked with Red Hat, TripAdvisor, MIT Lincoln Lab, HP Labs, and Johns Hopkins Center for Talented Youth. His research interests include cybersecurity (side-channels, smartphones, offensive and defensive), networking (security, sensors, localization), algorithms (data synchronization, edits, file sharing) and error-correcting codes (rateless coding, feedback). He has been awarded the ECE Teaching Awards from BU in 2003 and 2013, a Kern fellowship from BU in 2012, the NSF CAREER from BU in 2002, and the Kuck Outstanding Thesis from UIUC in 2000.

David Starobinski (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Technion–Israel Institute of Technology, in 1999. He is a Professor of Electrical and Computer Engineering, Systems Engineering, and Computer Science with Boston University. He was a Visiting Postdoctoral Researcher with the EECS Department, UC Berkeley from 1999 to 2000, an invited Professor with EPFL from 2007 to 2008, and a Faculty Fellow with the U.S. DoT Volpe National Transportation Systems Center from 2014 to 2019. His research interests are in cybersecurity, wireless networking, and network economics.

He received the CAREER Award from the U.S. National Science Foundation in 2002, the Early Career Principal Investigator Award from the U.S. Department of Energy in 2004, the BU ECE Faculty Teaching Awards in 2010 and 2020, and the Best Paper Awards at the WiOpt 2010, IEEE CNS 2016, and IEEE ICBC 2020 conferences. He has been on the editorial boards of the IEEE Open Journal of the Communications Society, the IEEE Transactions on Information Forensics and Security, and the IEEE/ACM Transactions on Networking.