This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

Near-accurate Multiset Reconciliation

Lailong Luo, Deke Guo, Xiang Zhao, Jie Wu, Ori Rottenstreich, Xueshan Luo

Abstract—The mission of set reconciliation (also called set synchronization) is to identify those elements which appear only in exactly one of two given sets. In this paper, we extend the set reconciliation problem into three design rationales: (i) multiset support; (ii) near 100% reconciliation accuracy; (iii) communication-friendly and time-saving. These three rationales, if realized, will lead to unprecedented benefits for the set reconciliation paradigm. Generally, prior reconciliation methods are mainly designed for simple sets and thus remain inapplicable for multisets. The methods based on probabilistic data structures, e.g., the Counting Bloom Filter (CBF), support efficient representation and multiplicity queries. Based on these probabilistic data structures, approximate multiset reconciliation can be enabled. However, they often cannot achieve a statisfying accuracy, due to potential hash collisions. The reconciliations enabled by logs or lists incur high time-complexity and communication overhead. Therefore, existing reconciliation methods, fail to realize the three rationales simultaneously. To this end, we redesign Trie and Fenwick Tree (FT), to near-accurately represent and reconcile unsorted and sorted multisets, respectively. Moreover, to further reduce the communication overhead during the reconciliation process, we design a partial transmission strategy when exchanging two Tries or FTs. Comprehensive evaluations are conducted to quantify the performance of our proposals. The trace-driven evaluations demonstrate that Trie and FT achieve near-accurate multiset reconciliation, with 4.31 and 2.96 times faster than the CBF-based method, respectively. The simulations based on synthetic datasets further indicate that our proposals outperform the CBF-based method in terms of accuracy and communication overhead.

Index Terms—Multiset Reconciliation; Trie; Fenwick Tree; Counting Bloom Filter.

1 Introduction

Consider a pair of hosts $Host_A$ and $Host_B$, which hold the sets A and B, respectively. The goal of set reconciliation for $Host_A$ and $Host_B$ is to search out the different elements and thereby deduce the union $A \cup B$ of set A and set B. In fact, set reconciliation is a common and fundamental task in a variety of networking scenarios and distributed systems. For instance, in a distributed file system [1], files usually need to be duplicated for disaster recovery via set reconciliation. In peer-to-peer networks [2], any pair of peers only need to exchange those missing blocks of a file from each other. For wireless sensor networks [3], the sink node only needs to collect those unobserved results from other hosts. For gossip protocol [4], any two nodes only need to exchange the different elements to save bandwidth and accelerate the convergence. In cloud computing applications, local devices (smartphones, laptops, robotics, and wearable equipments) only upload or download the nonexistent data from the Cloud [5].

The major challenge of set reconciliation is searching out the different elements between two sets, and then exchanging them in an accurate as well as fast fashion. A brute force method for set reconciliation is to transmit all elements in $Host_A$ to $Host_B$ with their multiplicities, and vice versa. This method is absolutely not advisable due to the huge cost of bandwidth. Alternately, a possible way is maintaining a log mechanism with timestamps in each host to record all the update events. Thereafter, whenever the hosts communicate again, the updated data can be synchronized. However, as stated in [6], the log mechanism has its inherent shortcomings, including requirement of system-level alterations, redundancy of hot items, lack of scalability, and strict requirement of networking and storage environment. Both the brute force method and the log-based solution are not bandwidth-friendly, since common elements may be exchanged between the hosts.

Therefore, several differential reconciliation techniques are proposed to identify and then transmit the different elements between set A and set B. Usually, the elements in a set are represented as a list. After exchanging the lists, $Host_A$ to $Host_B$ are able to determine and thereafter transmit the different elements. The complexity to distinguish different elements from common ones depends on the employed data structure. Consider a hash table as an example, $Host_A$ has to query all elements in A against the hash table from $Host_B$, and vice versa. Thus, the time-complexity is $O(n_A+n_B)$, where n_A and n_B are the number of elements in A and B, respectively. To further reduce the communication overhead, Bloom filter and its variants are employed to represent elements [6] [7] [8] [9] [10]. The Bloom filter, in contrast to the above, is space-efficient but probabilistic. As a consequence, part of the different elements may not be reconciled due to the potential false positive errors of the data structure.

There are also several exact approaches based on the theory of coding techniques. An error-correcting code based method has been developed for this problem with nearly optimal communication overhead by ranking string ele-

- *Lailong Luo, Deke Guo, Xiang Zhao and Xueshan Luo are with the Science and Technology Laboratory on Information Systems Engineering, National University of Defense Technology, Hunan, 410073, China. E-mail: {luolailong09, dekeguo, xiangzhao, xsluo}@nudt.edu.cn.*
- *Jie Wu is with the Department of Computer and Information Science, College of Science and Technology, Temple University, Philadelphia, Pennsylvania, USA. E-mail: jiewu @ temple.edu*
- *Ori Rottenstreich is with Orbs Research, Israel. E-mail: ori@orbs.com.*

  Corresponding author: Deke Guo.

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

ments in some order [11] [12] [13]. By evaluating the rational function of the characteristic polynomials, the different (not in common) strings can be decoded. This method can also reconcile discrete random variables, by combining the error-correcting mechanism with graph-coloring theories [14]. Unfortunately, this kind of solutions requires not less than $O(d^3)$ time to decode the difference, where d is the number of different elements between the sets. Besides, the difference estimators for these proposals are costly to implement.

Consequently, in this paper, we envision the following three design rationales for set reconciliation: (1) multiset support - the reconciled sets can be multisets which allow elements to have multiple replicas; (2) near-accurate - almost all the different elements will be identified; (3) communication-friendly and time-saving - the incurred communication overhead during reconciliation process is acceptable and the time-consumption is short. This vision, if realized, will lead to unprecedented benefits for set reconciliation. Firstly, we generalize the set reconciliation to multiset scenarios where existing reconciliation methods fail to function well. Secondly, near-accurate reconciliation ensures the QoS of the associated applications, since nearly all different elements will be identified and reconciled. Moreover, many applications need to invoke the reconciliation process frequently; hence, it is of great importance to reduce the bandwidth consumption during each round of reconciliation, especially for bandwidth-scarce situations. The time-saving characteristic further enhances the instantaneity of the methods.

Existing methods, however, fail to achieve the three rationales simultaneously. In fact, multiset generalizes the notion of a set and has been widely employed in a variety of distributed systems. Besides the general fields like philosophy, Logic, Linguistics, and Physics, multiset has also found its applications in mathematics and computer science [15]. Multisets has been applied in a variety of search and sort procedures. In communication networks, flows can be abstracted as a multiset by regarding a flow as an element and letting the number of packets the flow contains be the multiplicity. By gathering the flows together, a monitor mechanism is achieved reasonably [16]. Generally, in distributed scenarios, when two hosts need to maintain the same content, the multiset reconciliation problem arises.

To settle the multiset reconciliation and achieve the design rationales, we classify multisets into two categories, i.e., unsorted or sorted multiset. In an unsorted multiset, there is no strict constraint on the order of each element. Indeed, unsorted multiset is the general case for multiset. By contrast, in some special cases, an element is associated with its location in the set, e.g., the time-dependent variables, datasets with dedicated rules like the distance, similarity, or dictionary order. This implies an order on the elements of the set. Owning to such differentiation, we tackle the reconciliation of unsorted multisets and sorted multisets separately.

To reconcile two unsorted multisets, we first employ the tree-like data structure Trie [17] to represent each multiset by recording element information. Note that the traditional Trie can only represent a simple set, and fails to record the multiplicity of elements. Hence, we redesign the Trie by extending its nodes with additional information. Likewise, we further represent and reconcile two sorted multisets by redesigning Fenwick Tree (FT) [18], with respect of the three design rationales. Moreover, we design a partial transmission strategy for Tries and FTs to further reduce the communication overhead, during the reconciliation process. The basic idea is to summarize a multiset with a tree-like structure and prune identical subtrees that correspond to identical subsets of elements. After such pruning, the remained nodes only record the different elements.

TABLE 1
Notations and definitions

Notation	Definition
A, B	The two input multisets to be reconciled
A*, B*	The root sets of A and B, respectively
n_A, n_B	Number of elements in A* and B*, respectively
$m_A(x)$	Multiplicity of element x in A
D_E	Elements only exist in either A or B
D_{E_A}, D_{E_B}	Elements only exist in A and B, respectively
D_M	Common elements, yet with diverse multiplicities
D_{M_A}	Elements such that $m_A(x) > m_B(x)$
D_{M_B}	Elements such that $m_B(x) > m_A(x)$
D	The multiset denotes the different elements between A and B
d	The number of distinct elements in D
d_E, d_M	Number of elements in D_E and D_M, respectively
E_A, E_B	Elements transmitted between $host_A$ and $host_B$
r	Ratio between d_E and d
L	Number of levels in the tree data structure
$\mathcal{I}$	Number of bits for an element identifier
$\mathcal{C}$	Number of bits an element contains

The major contributions are summarized as follows:

- We formulate the set reconciliation problem and extend this problem into three design rationales, i.e., multiset support, near-accurate, and communication-friendly and time-saving. To achieve the three design rationales simultaneously, we redesign Trie and FT to near accurately represent and reconcile multisets.
- To further decrease the caused communication overhead, we investigate the partial transmission strategy when exchanging Tries and FTs between hosts. Specifically, by transmitting the Trie or FT part by part, the identical subtrees can be recognized and pruned and thus unnecessary transmission is avoided.
- Comprehensive experiments are conducted to measure the performance of our proposals, in terms of accuracy, communication overhead, and time-consumption. The trace-driven evaluations demonstrate that Trie and FT achieve near-accurate multiset reconciliation, with 4.31 and 2.96 times faster than the CBF-based method, respectively. The simulations based on synthetic datasets further indicate that our proposals outperform the CBF-based method in terms of accuracy and communication overhead.

The remainder of this paper is organized as follows. Section 2 briefly introduces several related data structures and formulates the multiset reconciliation problem. Section 3 describes how Trie and FT near-accurately represent and reconcile a pair of unsorted multisets and sorted multisets. The communication overhead of our reconciliation methods is quantified in Section 4. Section 5 presents the comprehensive evaluations. Lastly, we discuss several related issues in Section 6 and conclude this paper in Section 7.

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

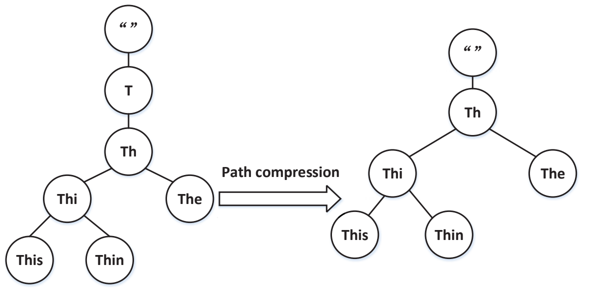


Fig. 1. A toy example of Trie which records three strings, i.e., “This”, “Thin” and “The”.

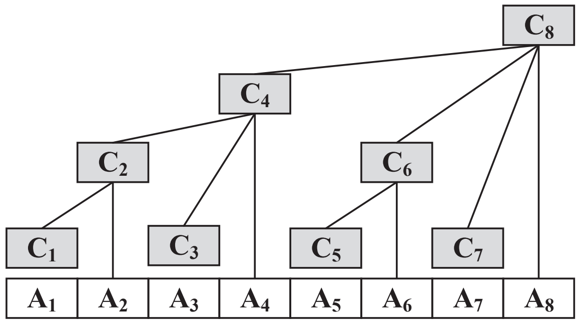


Fig. 2. An illustrative example of Fenwick Tree.

2 Preliminaries and Formulations

In this section, we start with the basic concept of multiset followed by the three kinds of data structures that are employed to realize multiset reconciliation. Thereafter, we formulate the multiset reconciliation problem. Table. 1 summarizes the notations and their definitions in this paper.

2.1 Prior knowledge

2.1.1 Multiset

Unlike a simple set, a multiset allows elements to appear for multiple times [15]. For clarity, several parameters are employed to describe a multiset. Let x be an element of a multiset A, then the multiplicity of x, denoted as $m_A(x)$, is utilized to record the number of instances of x in A. Besides, a simple set A* is defined as the root set [15] of A such that A*$=\{x \in A | m_A(x) > 0\}$. A root set can be shared by different multisets associated with different element multiplicities. A multiset A can be characterized as its root set and the multiplicity of each element. That is, a multiset A can be represented as a set of pairs like $A=\{\langle x_1, m_A(x_1)\rangle, \cdots, \langle x_i, m_A(x_i)\rangle, \cdots\}$. The union of two multisets A and B is to calculate a new multiset U such that $m_U(x) = \max\{m_A(x), m_B(x)\}$. Correspondingly, the intersection between two multisets A and B is a new multiset I such that $m_I(x) = \min\{m_A(x), m_B(x)\}$.

2.1.2 Counting Bloom filter

As a typical probabilistic data structure, Counting Bloom Filter (CBF) [9] is a well-known variant of the traditional Bloom filter [8]. Bloom filter (BF) [8] represents n elements with a bit vector of length m. All of the m bits in the vector are initially set to 0. A group of k independent hash functions, $\{h_1, h_2, ..., h_k\}$, are employed to randomly map each element into k positions, $\{h_1(x), h_2(x), ..., h_k(x)\}$, in the bit vector. For each set element, those bits at such k positions in the vector are all set to 1. Upon querying, if any bit at the k hashed positions of the element equals 0, BF judges that x does not belong to the set. CBF replaces each bit in the BF vector with a counter consisting of multiple bits [9]. In this way, it naturally supports deletions and insertions of elements. The value of each counter can exceed 1. Assume an element x is hashed into the 4^{th}, 10^{th}, and 15^{th} counters, while element y is hashed into the 5^{th}, 15^{th}, and 24^{th} counters, respectively. Consequently, the value is 1 for the 4^{th}, 5^{th}, 10^{th} and 24^{th} counters, but is 2 for the 15^{th} counter. When the element x is deleted from the set, the values of the 4^{th}, 10^{th}, and 15^{th} counters are decreased by 1, but the value of the 15^{th} counter still remains positive (with a value of 2-1=1). That is, the membership information of the element y is still kept in the updated CBF.

2.1.3 Trie

Initially, Trie was proposed to store strings [17] [19]. Thereafter it has been widely employed in the area of routing, information retrieval, image segmentation, geographic information systems, and even robotics. Basically, each string is represented by a leaf in the Trie, and the value of the string is the path from the root to this leaf. Trie is constructed based on the prefix of the strings. Recursively, all strings with a common prefix share a common node in the Trie. For example, consider three strings, “This”, “Thin” and “The”. As shown in Fig. 1, since they share a common prefix “T”, these strings have the same root node in the generated Trie. Similarly, “This” and “Thin” will share the same node “Thi”. Note that, Trie may be inefficient due to its large amount of nodes and long average depth. Hence, in real use, the *path compression* technique [20] is utilized to prune the interval nodes with only one child (an example is also given in Fig. 1). Another alternative solution is the so-called *level compression* technique [21], which replaces the i^{th} highest complete levels of a binary Trie (representing binary strings) with a single node of degree 2^i. In this paper, we employ the *path compression* technique by default.

2.1.4 Fenwick Tree

Fenwick Tree (FT) is a data structure mostly employed for representing an array of numbers and calculating a prefix sum with the time-complexity of $O(\log n)$, where n is the number of elements recorded by FT [18]. Adding an element to an arbitrary position in the FT costs $O(\log n)$ [22]. As depicted in Fig. 2, a FT contains both Carry nodes and Array nodes, denoted as C_i and A_i, respectively. Indeed, a Carry node is responsible for recording the information of a dedicated number of elements in the array. The content in C_i can be calculated by the equation $C_i = \sum_{k=i-2^{\sigma_i}+1}^{i} A_k$, where σ_i is the number of consecutive zero bit in the suffix of a binary representation of i. For instance, 4 is represented as 100 in the binary system and σ_4=2, thus we have $C_4 = A_1 + A_2 + A_3 + A_4$. In other words, C_4 stands for the first 4 elements in the array. Note that the number of Array nodes in FT is exactly the same as the number of elements in the array (for instance there are eight nodes of each kind in Fig. 2). Besides, for any C_i has σ_i+1 children and the locations of these children can be derived as $C_{i-2^{j-1}}$, where $j \in [0, \sigma_i]$. Usually, FT is employed to resemble binomial trees and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

therefore enables manipulations towards binomial queues [23], since it provides lower maintenance cost (less storage, more conceptional simplicity, lower time-complexity) than previous proposals such as 2-3 trees [24], leftist trees [25].

2.2 Problem Formulation

Consider that two hosts, $Host_A$ and $Host_B$, keep two multisets, A and B, respectively. Reconciling A and B means searching out and thereafter exchanging the different elements, such that the multisets are updated in each of the hosts till finally $A{=}B$. Prior reconciliation methods for simple sets, however, remain inapplicable for the multiset reconciliation. The reason is that, the difference between two multisets stems from two sources. The first kind of difference is D_E, which denotes the elements only exist in either A or B, i.e., $D_E{=}\{x \in A \cup B | x \notin A \ or \ x \notin B\}$. Specifically, D_{E_A} denotes those elements that only exist in set A, while D_{E_B} denotes those elements that only exist in set B. Thus, we have $D_E{=}D_{E_A} \cup D_{E_B}$. The second kind of difference, denoted as D_M, includes the elements which appear in both A and B, but have distinct multiplicities, i.e., $D_M{=}\{x{\in}A^*{\cap}B^* \ \ |m_{D_M}(x){=}|m_A(x){-}m_B(x)|\}$. Typically, D_{M_A} denotes the common elements, whose multiplicities in A are larger than that in B. Meanwhile, D_{M_B} denotes the common elements, whose multiplicities in A are smaller than that in B. Thus, we have $D_M{=}D_{M_A} \cup D_{M_B}$. By contrast, from the view of hosts, let $D_A{=}D_{E_A} \cup D_{M_A}$ denote the different elements come from A, and $D_B{=}D_{E_B} \cup D_{M_B}$ denote the different elements caused by B. Then we have $D{=}D_A \cup D_B{=}D_E \cup D_M$.

To resolve the multiset reconciliation, we investigate a reconciliation framework which contains the encoding, subtracting, and decoding operations of involved mulitsets and a series of general steps. The encoding, subtracting, and decoding operations depend on the utilized data structures. Thus, such operations may vary under different reconciliation methods. Let E_A (E_B) represent the elements sent from $Host_A$ ($Host_B$) to $Host_B$ ($Host_A$) during the reconciliation. The following general steps, are indispensable in accomplishing fast and efficient multiset reconciliation.

1) Each host executes the encoding operation, which stores the information of each element in its multiset with a dedicated data structure.
2) $Host_B$ sends the encoding result of multiset B to $Host_A$.
3) Given the encoding results of the two multisets, $Host_A$ executes the subtracting and decoding operations to derive those elements, which need to be transmitted to $Host_B$, i.e., E_A.
4) $Host_A$ sends the subtracting result of the two encoding results together with E_A, to $Host_B$.
5) Once receiving content from $Host_A$, $Host_B$ then executes the decoding operation to identify those elements in E_B, which need to be sent to $Host_A$.
6) $Host_B$ sends E_B to $Host_A$ and the reconciliation process is accomplished.

Ideally, an efficient multiset reconciliation method should only exchange those elements in D_E's root set, i.e., $E_A{=}(D_{E_A})^*$ and $E_B{=}(D_{E_B})^*$. That is, the elements in D_E will be transmitted only once, and the host will generate dedicated number of replicas of these elements such that $m_A(x){=}m_B(x)$ for all $x{\in}D_E$. Certainly, the elements in D_M will also be reconciled by generating replicas in each host, rather than transmitting the elements directly. By doing so, the reconciliation spends the least bandwidth. Therefore, in this paper, we present the reconciliation methods based on the redesigned Trie and FT. With our proposals, to save bandwidth we try to avoid transmitting elements not in D_E.

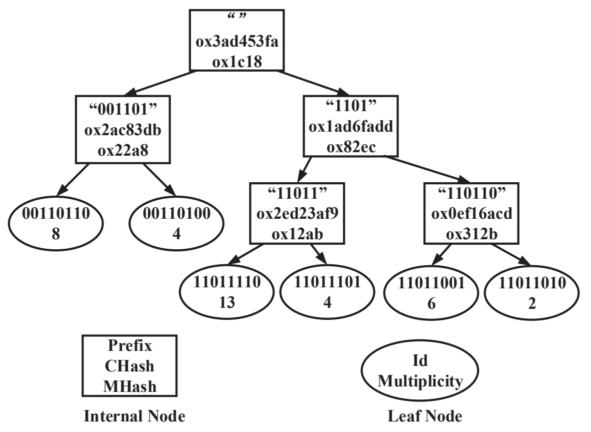


Fig. 3. Representing $\{\langle x_1, 8\rangle, \langle x_2, 4\rangle, \langle x_3, 13\rangle, \langle x_4, 4\rangle, \langle x_5, 6\rangle, \langle x_6, 2\rangle\}$ with the redesigned Trie.

3 Accurate Multiset Reconciliation

In this section, we consider the multiset reconciliation problem for achieving the design rationales. We find that imposing an order to all elements in a multiset is helpful in deriving those different elements during the reconciliation process. Therefore, we redesign the Trie data structure to impose an order to the elements in unsorted multiset based on their Ids. As for sorted multisets, the elements are already given with a dedicated order. Thus, we redesign the FT to establish an index tree without changing the given order.

3.1 Representation of unsorted multisets

A challenging issue when representing an unsorted multiset for reconciliation is that the employed data structure should impose an order to all elements such that the different elements between two multisets can be easily located. Fortunately, a Trie naturally supports the ordering of elements, since it constructs an index tree based on the Ids of all elements it represents.

However, to represent a multiset, each node in a traditional Trie needs more information to record the multiplicity of each covered element. As depicted in Fig. 3, the augmented Trie contains two kinds of nodes, i.e., internal nodes and leaf nodes. Internal nodes are responsible for constructing the Trie by keeping the aggregated information. As for the leaves, they record the basic information of each element in the multiset with Id and $Multiplicity$. The Id field of a leaf node is a fixed-length binary string, which uniquely identifies this element. The $Multiplicity$ field counts the multiplicity of the element.

Each internal node consists of three fields: *Prefix*, *CHash* and *MHash*. The *Prefix* represents the common bits in the

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1 Subtracting Tries at $Host_A$

```
Require: The generated $Trie_A$ and $Trie_B$, the number of bits of each identifier ($\mathcal{I}$).
1: for $i$=0 to $\mathcal{I}-1$ do
2:     Let $Location$ be a vector of integers;
3:     $Location$ = LocateSameNode($Trie_A$, $Trie_B$, $i$);
4:     % The locations of the same node in $Trie_A$ and $Trie_B$ are saved in the even and odd positions of $Location$, respectively.
5:     for $j = 0$ to $Location.size()/2-1$ do
6:         $Trie_A$ = PruneTrie ($Trie_A$, $Location[2j]$);
7:         $Trie_B$ = PruneTrie ($Trie_B$, $Location[2j{+}1]$);
8: return $Trie_A$ and $Trie_B$;
```

Ids of elements covered by this internal node. We notice that the same prefix may be shared by diverse combinations of elements. Hence, the *CHash* field and *MHamlsh* field are employed to further represent the subtrie roots from this node. To calculate the *CHash*, we first combine the prefix or *CHash* of its children together and then hash the combination value to a unique number. For example, in Fig. 3, left and right child of node "11011" are 11011110 and 11011101 respectively. We combine them together and derive the combination as 1101111011011101. Note that, when calculating the combination, we put the left child before the right one. There is no strict requirement for the order of its children, but for consistency one order should be employed through out the algorithm. A hash function maps the combination to be a unique number, i.e., ox2ed23af9 in the above example. As a result, the *Prefix* and *CHash* together can determine the structure of the subtrie that roots from this node. In other words, if two internal nodes have the same *Prefix* and *CHash*, then all elements in the related subtrie are the same with high probability.

Additionally, each internal node needs a field to represent the multiplicities of the elements in the subtrie. Note that, simply adding the multiplicity of all its children together cannot accurately identify the $Multiplicity$ of its children, since different combinations of integers may lead to the same sum result. Hence, we investigate the *MHash* field to exclusively denote the $Multiplicity$ of the elements in the subtrie. To calculate the *MHash*, just like the calculation of $CHash$ we first combine the *MHash* of its children together and then hash the combination value to a unique number. For example, in Fig. 3,

To establish our redesigned Trie, three steps are needed. Firstly, following its definition, we build the basic structure of the Trie based on the Id of each element. Secondly, we delete unnecessary internal nodes by employing the *path compress algorithm* [20]. After compression, only $n-1$ internal nodes and n leaves remain in the resultant Trie. Thirdly, we complete the fields of each internal node in a bottom-up manner. In this step, the *MHash* and *CHash* of each internal node are derived by combining the *MHash* and *CHash* of its children together. In this way, the *Prefix* helps to build the Trie, and the *CHash* and *MHash* exclusively identify the elements and their multiplicities in the subtrie. When reconciling, two multisets A and B are represented as $Trie_A$ and $Trie_B$, respectively. By traversing $Trie_A$ and $Trie_B$, our reconciliation method can prune the pair of subtries if their nodes are exactly the same. The reconciliation method avoids comparing nodes within the pruned subtries, assuming they are identical for the two sets.

3.2 Reconciliation of unsorted multisets

Following the six steps presented in Section 2.2, once receiving $Trie_B$ from $Host_B$, $Host_A$ compares $Trie_A$ and $Trie_B$ to identify those elements in D_{E_A} and D_{M_A}. Similarly, $Host_B$ will derive D_{E_B} and D_{M_B} by comparing $Trie_A$ and $Trie_B$. After exchanging the diverse elements in D_{E_A} and D_{E_B}, the reconciliation will be accomplished.

Encoding. The encoding operation for an unsorted multiset is to represent a multiset by correctly building a redesigned Trie. Given the Id of each element, according to the definition, a Trie for a multiset can be easily constructed. Note that the same set of hash functions should be employed when calculating the Id of each element at $Host_A$ and $Host_B$. An identical element at $Host_A$ and $Host_B$ will result in the same Id. Besides, to lessen the potential hash conflicts, the length of Id should be long enough.

Subtracting. The mission of subtracting at each host is to prune the pairs of subtries which stand for the same set of elements. As a result, only those different elements will remain in the Tries. Generally, two schemes are available to traverse the Tries, i.e., the breadth-first search (BFS) and the depth-first search (DFS). Note that, the BFS scheme can prune the subtries early. By contrast, the DFS scheme will locate the first different element quickly, allowing to transmit the different elements early. We employ the BFS scheme since it prunes the subtries more quickly.

Taking the subtraction at $Host_A$ as an example, the detail is given in Algorithm 1. Note that, if an Id consists of $\mathcal{I}$ bits, then the Trie includes at most $\mathcal{I}+1$ levels of nodes. The BFS scheme searches the Trie level by level. In each level, the algorithm first calculates the locations of those same nodes in the two Tries (Line 3). The vector $Location$ records the locations of such pairs in $Trie_A$ and $Trie_B$, respectively. Then we prune those same nodes in both $Trie_A$ and $Trie_B$. Eventually, the resultant $Trie_A$ and $Trie_B$ just represent the different elements.

Decoding. After pruning those same elements, $Host_A$ and $Host_B$ decode the remaining Tries to derive the different elements. Since the Id in each leaf node identifies the element, the decoding operation can be realized by comparing the Id part of the left leaves. Note that by traversing the Tries, we cannot only search out the difference between the two multisets, but also distinguish D_E from D_M. For any pair of internal nodes, if their *Prefix* and *CHash* are the same but have diverse *MHash*, those elements covered by the two nodes belong to D_M. On the contrary, if neither *Prefix* nor *CHash* is the same, then at least one of the elements covered by the two nodes belongs to D_E. Differentiating D_E and D_M is of great significance in decreasing the communication overhead, since only those elements in D_E are required to be sent to the other host. As for those elements in D_M, the host just needs to generate given numbers of replicas. For example, if $m_A(x)$=3 and $m_B(x)$=5, $Host_A$ just needs to derive 2 additional instances of x, instead of receiving two copies of x from $Host_B$.

With the above operations and following the steps in Section 2.2, Trie supports the reconciliation of two unsorted

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

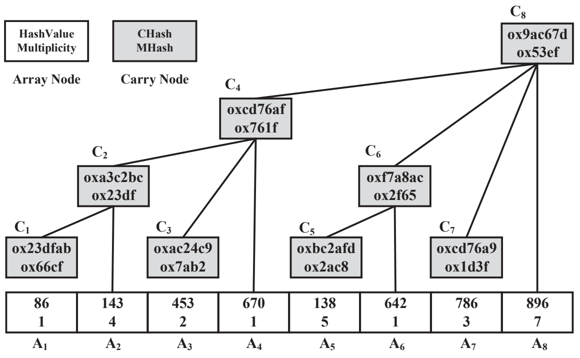


Fig. 4. A toy example of representing a sorted multiset with FT.

multisets reasonably. During the reconciliation, the two Tries should be transmitted to derive the different elements. The transmitting strategy is later introduced in Section 3.5.

3.3 Representation of sorted multisets

The redesigned Trie structure for an unsorted multiset, however, is not necessary for a sorted multiset. The reason is that all elements have already formed a given order in a sorted multiset. For this reason, we redesign the Fenwick Tree (FT) to represent and reconcile sorted multisets near accurately.

As shown in Fig. 4, an FT consists of two kinds of nodes, i.e., the Carry nodes and the Array nodes. The Carry nodes are responsible for recording the aggregation information of the elements they represent, and Array nodes store the raw information of each element. Each Array node includes two fields, i.e., the $hashValue$ that identifies each element, and the $Multiplicity$ field that counts the number of instances of an element. The $hashValue$ is generated by hashing the element to a specific interval, e.g., $[0, 2^{20}-1]$. As for each Carry node, it consists of the *CHash* and *MHash* fields. Similar with the internal nodes in Trie, the *CHash* is also calculated by hashing the combination of the $hashValue$ or *CHash* of all its children. The *MHash* field is derived by hashing the combination of its children's *MHash* or $Multiplicity$ to be an integer. In this way, a sorted multiset can be represented by our redesigned FT.

When comparing two FTs, the *CHash* field judges whether the two subtrees stand for the same elements, and the *MHash* field tells if they have the same multiplicity. The different elements will be reflected in the *CHash* fields. Therefore, to reconcile sorted A and B, $Host_A$ and $Host_B$ establish and exchange FT_A and FT_B, respectively. During traversing, the same subtrees in FT_A and FT_B are pruned and thereby exposing the different elements. The specific operations will be detailed thereafter.

3.4 Reconciliation of sorted multisets

In this section, we begin our proposal with the design of encoding, subtracting and decoding operations for FTs. Since the elements are pre-ranked, we only need to aggregate the elements together to speed-up the reconciliation processes.

Encoding. The encoding operation means to represent a multiset as an FT structure. This encoding task is simple since the children of each Carry node are determined and can be calculated directly. To ensure that the same elements have the same $hashValue$, $Host_A$ and $Host_B$ should employ the same hash function to derive the unique fingerprint of each element.

Algorithm 2 Subtracting FTs at $Host_A$

```
Require: The generated $FT_A$ and $FT_B$, the number of levels in
    the FT $L$.
1: for $i$=0 to $L-1$ do
2:     Let $Location$ be a vector of integers;
3:     $Location$ = LocateSameNode($FT_A$, $FT_B$, $i$);
4:     % The locations of the same node in $Trie_A$ and $Trie_B$
       are saved in the even and odd positions of $Location$,
       respectively.
5:     for $j$ = 0 to $Location.size()/2-1$ do
6:         $FT_A$ = PruneTrie ($FT_A$, $Location[2j]$);
7:         $FT_B$ = PruneTrie ($FT_B$, $Location[2j+1]$);
8: return $FT_A$ and $FT_B$;
```

Subtracting. As stated in section 2.2, after exchanging the resulted FTs, the two hosts will traverse both FT_A and FT_B to distinguish the different elements. Similar with the traverse algorithm used by Tries, we employ the BFS scheme to compare the nodes in FTs. In this way, the algorithm will prune the same subtrees rapidly. Without loss of generality, we elaborate the subtracting operation at $Host_A$. As illustrated in Algorithm 2, if the maximum number of levels in FT_A and FT_B is L, the algorithm will be repeatedly executed for L rounds. In each round, we first find out the same nodes and denote them in the integer vector $Location$ (Line 3). Then, for each same pair of nodes, we prune the same subtree from FT_A and FT_B. For any pair of Carry nodes, if they share the same *CHash* and *MHash*, we believe that they represent the same elements. Otherwise, they have at least one different element. After pruning, the remained elements in FT_A and FT_B are just those elements in D_A and D_B.

Decoding. After the subtracting operations, the resultant FT_A and FT_B only contain those elements in D. By judging the $hashValue$ and the $Multiplicity$ of the left Array nodes, the reconciliation algorithm will recognize those different elements. At last, $Host_A$ and $Host_B$ exchange the distinct elements in D_A and D_B, and the reconciliation process is completed. Note that, by comparing the nodes in FTs, the reconciliation algorithm can also distinguish D_E from D_M. Specifically, if two nodes share the same *CHash* but differ in the *MHash* field, these two nodes only cover elements in D_M. If the *CHash* fields are distinct, these two nodes must cover at least one element belongs to D_E. Also, to save bandwidth and shorten reconciliation time, only the elements in D_E should be transmitted. The elements in D_M will be reconciled via producing replicas at each host.

By executing the designed operations and following the reconciliation steps, the redesigned FT enables the sorted multiset reconciliation. Also, the FTs need to be exchanged between the two hosts. To further save bandwidth, the partial transmission strategy is introduced.

3.5 Bandwidth-saving exchanging

As stated above, the redesigned Trie and FT are employed to represent and reconcile multisets near-accurately. During the reconciliation process, $Host_A$ and $Host_B$ exchange

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

$Trie_A$ and $Trie_B$ (or FT_A and FT_B), to distinguish the different elements in multisets A and B. A straightforward way is to transmit the whole data structure to the other host directly. This method, however, leads to inefficient communication, since it transmits all nodes in the associated data structures. Typically, it is unacceptable in bandwidth-scarce networks.

Fortunately, we notice that the nodes in Trie or FT can be transmitted one by one or part by part. By determining whether the nodes in the subtrie or subtree should be transmitted, unnecessary communication overhead can be avoided. Consider the situation that $Host_A$ sends an internal node $Trie_A[i]$ to $Host_B$. Upon receiving $Trie_A[i]$, $Host_B$ searches $Trie_B$ with the BFS searching scheme and decides whether there exists a node that is exactly the same with $Trie_A[i]$. If the result is true, it means both A and B have the same elements, which are represented by the subtrie rooting from $Trie_A[i]$. Hence, the nodes in the subtrie rooted from $Trie_A[i]$ will be deleted from both $Trie_A$ and $Trie_B$. Note that, $Host_A$ will send the root node first, then the internal nodes in level 1, thereafter internal nodes in level 2, so on and so forth. In this order, the same subtries will be pruned and the underlying nodes will not be exchanged anymore. Similarly, FT will also benefit from this strategy. To enable the least amount of communication overhead, the internal nodes or Carry nodes covering more elements will be transmitted first.

Definitely, the partial transmitting method results in multiple rounds of communication. However, we argue that this strategy has no severe requirement of bandwidth, which is of great importance for opportunistic or bandwidth-scarce networks, e.g., Internet of Vehicles and Wireless Sensor Networks. Employing the partial transmission strategy or not, depends on the users' network circumstances and their requirement of reconciliation delay.

4 Analysis of Communication Overhead

Note that, the communication overhead of multiset reconciliation consists of two components, i.e., the exchanging of the employed data structure and the transmission of different elements. As stated in Section 2.2, the reconciliation methods should only transmit the elements in D_E for once. Therefore, this kind of communication overhead is determined by D_E and thus has merely optimization space. Especially, when the two multisets are totally disjoint, all the elements in both A and B should be transmitted. Therefore, the lower bound of transmitting elements can be formulated as $(n_A{+}n_B){\cdot}(\mathcal{I}{+}\mathcal{C})$, where n_A and n_B are the number of elements in root sets A^* and B^*, while $\mathcal{I}$ and $\mathcal{C}$ are the number of bits for each element in the employed data structure and the maximum number of bits to express an element's content. In practice, there are little things we can do to reduce the transmission of elements. Consequently, we focus on optimizing the exchanging of the employed data structures.

Typically, our methods exchange the generated FT or Trie between the two hosts to derive the different elements. Definitely, the FTs or Tries can be transmitted entirely in one communication round, with communication overhead of $O(n)$, where n is the number of elements in the multiset.

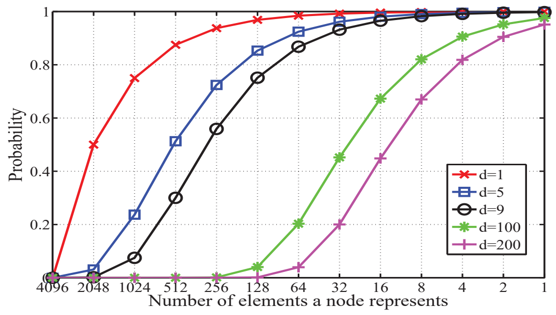


Fig. 5. The relationship between the probability that a node aggregates no information of elements in D and the number of elements covered by this node (both Trie and FT), with given the number of elements in a multiset as n=4096.

Specifically, there are $2n{-}1$ and $2n$ nodes in Trie and FT. Consider that each node has a constant number of bits, the total communication overhead can be approximate as $O(n)$. Note that, the transmission overhead of both Trie and FT will not be affected by the total cardinality of the involved elements, or the real volume of each element, but only determined by the number of elements in the root set. That is, the transmission cost is robust when the root sets are given. Alternatively, as stated in Section 3.5, the nodes in both Trie and FT can be transmitted with a partial manner. We suppose that the elements in D_A and D_B are randomly distributed in the derived Trie and FT, and an element in D_E will not appear in D_M. With this assumption, we analyze the communication overhead of the partial transmission strategy as follows.

4.1 Communication overhead of exchanging Tries

The communication overhead depends on the probability that two internal nodes are the same in the resulted Tries and FTs. In this section, w.l.o.g, we consider $Trie_A$ as an example. Let $E_A(T[i])$ denote the elements that rooted from an internal node $Trie_A[i]$, where $i{\in}[1, n{-}1]$. Moreover, $n_T[i]$ records the number of elements in $E_A(T[i])$, and α_i counts the number of elements in the set $E_A(T[i]) \cap D_A$. In other words, α_i denotes the number of different elements that covered by the subtrie rooted from the internal node $Trie_A[i]$. Consequently, α_i=0 means the subtrie rooted from $Trie_A[i]$ covers no different elements. Then we have:

$$p(\alpha_i{=}0) = \frac{\binom{n-d_A}{n_T[i]}}{\binom{n}{n_T[i]}}, \tag{1}$$

where n is the number of elements in the multiset's root set. Basically, $\binom{n-d_A}{n_T[i]}$ implies choosing $n_T[i]$ elements from the $n{-}d_A$ elements in $A \cap B$.

As depicted in Fig. 5, we note that, with the increase of d, the nodes in a Trie are more likely to record the information of elements in D. By contrast, with given n and d, the nodes that represented more elements have a higher probability of including an element in D. We conclude that higher level of nodes will be transmitted with a higher probability during the reconciliation process. That is, given a Trie, the probability that an internal node should be transmitted is related to the number of different elements in its subtrie.

After calculating the probability of α_i=0, then according to the structure of the established Trie, we can derive the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

number of elements that each internal node stands for. Hence, the expectation of communication overhead can be calculated as follows:

$$C_T = b_T \times \sum_{i=1}^{n-1} (1-p(\alpha_i=0)), \quad (2)$$

where b_T is the number of bits for each internal node. In practice, the exact communication overhead depends on the distribution of the different elements in the Trie. For example, if the different elements are neighbouring with each other in the Trie leaves, then the caused communication overhead is low. By contrast, when the different elements are fully dispersed among the leaves, more internal nodes should be transmitted. Surely, when the number of different elements are large enough, all the internal nodes have to be transmitted. Theoretically, when $d=n/2$, it is possible we need to transmit the whole Trie.

4.2 Communication overhead of exchanging FTs

Given a Carry node $F[i]$ in an established FT called FT_A, let $E_A(F[i])$ denote the elements represented by the subtree that rooted from $F[i]$, where $i\in[1,n]$. Moreover, $n_F[i]$ records the number of elements in $E_A(F[i])$, and β_i counts the number of elements in the set $E_A(F[i]) \cap D_A$. Similar with α_i, $\beta_i=0$ implies the subtree contains none of the elements in D_A, and this Carry node is unnecessary to be transmitted to $Host_B$. With these notations, we have:

$$p(\beta_i=0) = \frac{\binom{n-d_A}{n_F[i]}}{\binom{n}{n_F[i]}}, \quad (3)$$

where n is the number of elements in the multiset's root set.

Fig. 5 also plots the relationship between $p(\beta_i=0)$ and the value of $n_F[i]$, as well as D_A. In fact, $p(\alpha_i=0)$ and $p(\beta_i=0)$ follow the same distribution function. That is, $p(\beta_i=0)$ is proportional to both $n_F[i]$ and d_A. Based on the calculated probability, the expectation of communication overhead can be determined as:

$$C_{FT} = b_{FT} \sum_{i=1}^{n} (1-p(\beta_i=0)), \quad (4)$$

where b_{FT} is the number of bits for each Carry node. FT is structurally different with Trie. As a result, given the same value of d, they may need to exchange diverse amount of nodes. Also, in the cases where all the elements in D are totally dispersed among the Array nodes, more Carry nodes should be transmitted than the case where the different elements are neighbouring with each other.

From the above analysis, the partial transmission strategy can significantly reduce the communication overhead to deduce the elements in D. Typically, larger d leads to higher communication overhead for both Trie and FT. It is possible that all the nodes in Trie and FT should be transmitted. However, the different scale between two multisets is usually small so that the communication overhead is acceptable. Moreover, when d is larger than $n/2$, we advice the users to exchange the Tries and FTs entirely. Since in that case, our partial transmission strategy may not save much communication overhead but consumes considerable reconciliation delay.

5 Evaluation

In this section, we implement the proposed methods to evaluate the reconciliation accuracy, communication overhead, and time-consumption. Especially, we measure our proposals based on the real CAIDA trace. Thereafter, we further quantify the reconciliation performance with synthetic datasets to specify the impact of each parameter.

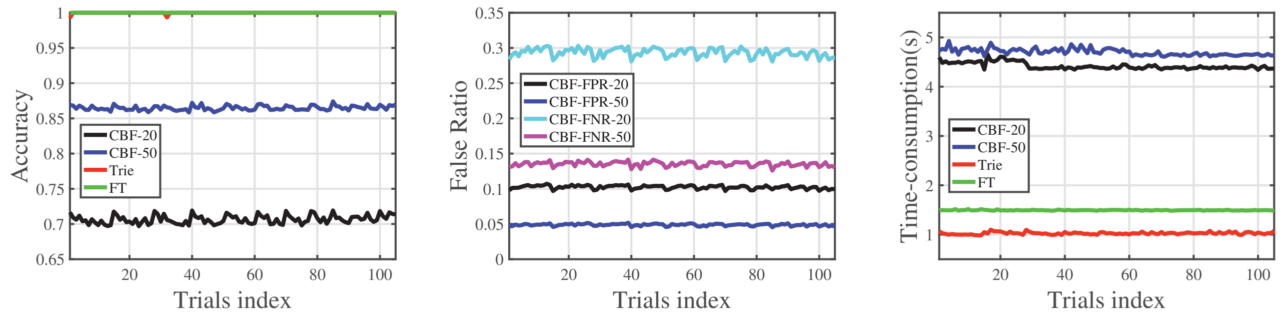
5.1 Experiment methodology

Implementation. In our experiments, a testbed with 2.5 GHz CPU and 8 GB RAM is employed as a host. We explore the impact of three varied parameters on the reconciliation performance, including the number of elements n, the size of difference between two multisets d, and the difference ratio $r=d_E/d$. Note that, r ranges from 0 to 1. Extremely, $r=0$ means all different elements stem from the diverse multiplicities of elements. By contrast, when all differences are caused by the diverse elements, $r=1$. For simplicity, in the following experiments, we set $n_A=n_B=n$, and the *CHash* and *MHash* fields are given as 30 bits.

Measurements. To quantify the three design rationales, we evaluate the performance of our proposals in terms of three measurements, i.e., reconciliation accuracy, communication overhead, and time-consumption, respectively. As a fundamental metric, reconciliation accuracy is defined as the ratio of the number of correctly identified elements (both the elements and their multiplicities should be correct) to the number of elements in D. Note that, bandwidth is usually a scarce resource in many networks. Hence, we compare the communication overhead of exchanging counters or nodes in CBFs, Tries, and FTs. Besides, time-consumption is a fatal metric for delay-sensitive or online applications. In our evaluation, we only quantify the time-consumption caused by the reconciliation process and don't take into account the delay due to the network latency. We report all the metrics on average after 100 rounds of tests.

Datasets. The employed CAIDA dataset (anonymized Internet traces 2016 dataset [26]) contains anonymized passive traffic traces from CAIDA's *equinix-chicago* monitor on high-speed Internet backbone links. We slice the dataset into 15 subsets based on the timestamps. For each subset, we extract the source IP, destination IP and protocol to label each packet. In this manner, we construct a multiset, in which an element is an Internet flow which has multiple packets with same source IP, destination IP and protocol, and the multiplicity is the number of the contained packets. Therefore, we have 15 trace-driven multisets. By reconciling these multisets pairwisely, we have 105 real trails. Additionally, we also generate synthetic multisets by adjusting the length and characters in strings to generate varied multisets. The multiplicity of an element is chosen randomly from a dedicated range of integer. We vary the total number of elements n, the number of different elements d and the difference ratio r (the ratio between d_E and d), and hence generate diverse multisets. By running the algorithms upon these synthetic multisets, we uncover the performance impact of these parameters.

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



(a) Accuracy. (b) The FPR and FNR of CBF. (c) Time-consumption.

Fig. 6. The trace-driven comparisons in terms of reconciliation accuracy and time-consumption. The values are calcualted for 105 trials.

5.2 Performance evaluation with real trace

In this section, we implement our proposals and compare them by reconciling real Internet traces. The results are depicted in Fig. 6. As shown in Fig. 6(a), both the Trie-based method and the FT-based method achieve near-accurate reconciliations. The reason for the failures of searching out a few different elements is the inherent hash collisions. We implement 30-bit *MHash* and *CHash* fields in both Tries and FTs. As a result, the probability of a collision-free *MHash* is $q_0{=}1{-}1/2^{30}$. Thereby, the probability that all *MHash* fields in a Trie which records n_A elements will be $q_0^{n_A}$. By jointly considering both *MHash* and *CHash*, the probability of a collision-free Trie is, therefore, $q_0^{2n_A}$. In our experiments, the value of n_A is at the level of 10^5. As a result, the reconciliation accuracy is around 0.9999. The FT-based method achieves a similar accuracy.

As for the CBF-based method, we set the counters per element (cpe) as 20 and 50 respectively. The resultant accuracy fluctuates around 0.70 and 0.86 respectively. Correspondingly, Fig. 6(b) records the false positive rate (FPR) and false negative rate (FNR) for the CBF-based method. Note that the quantified accuracy is right $1{-}FNR$ since the accuracy is defined as the ratio between the number of identified difference and the total number of real difference. Unlike the Trie-based method and the FT-based method, the CBF-based method fails to achieve near-accurate multiset reconciliation. Intuitively, the reconciliation accuracy of the CBF-based method can be improved at the cost of more space. The temporal distribution of the multisets leads to a diverse scale of difference between the two reconciled multisets. Usually, two temporal neighboring multisets may share more elements than two multisets with a large time interval. As a result, the accuracy of CBF-based method will be affected. Take the "CBF-FNR-20" in Fig. 6(b) as an example. The FNR increases from 0.283 in trail 1 (which reconciles multisets at time period 0 and time period 1) to 0.302 in trail 14 (which reconciles multisets at time period 0 and time period 14). Other curves in Fig. 6(b) also experience similar fluctuation.

Furthermore, Fig. 6(c) depicts the time-consumption for the proposed methods to search out the different elements. Generally, CBF needs more time to execute the subtracting operation and thereafter query the elements. Sure, with more counters introduced in, the subtracting will consume more time. This explains why "CBF-50" is more time-consuming than "CBF-20". Additionally, both the redesigned Trie and FT spend much less time to recognize the different elements than CBF. Trie even outperforms FT with 0.468 seconds less time-consumption on average. The reason is that the *Prefix* field in Trie identifies different internal nodes directly so that Trie needn't compare the *CHash* and *MHash* fields further. Generally, the *Prefix* has much fewer bits than the *CHash* and *MHash* fields. As a result, checking the *Prefix* is much faster than checking *CHash* and *MHash*. For CBF, the time-consumption is proportional to the number of elements in the multisets. Therefore, the curves "CBF-50" and "CBF-20" undulate accordingly. Trie and FT, in effect, cost more time when the difference is large. In our trails, the difference scale d varies, but the gaps among them are not sufficient to bring significant impact to the time-consumption of the redesigned Trie and FT.

For "CBF-20", the bits per element is $20{\cdot}4{=}80$. Similarly, the bits per element is 200 for "CBF-50". The Carry nodes in a redesigned FT consumes only 60 bits in our experiments. Therefore the bits per element is 60 since there are n Carry nodes to represent n multiset elements. As for the redesigned Trie, there are $n{-}1$ internal nodes each of which costs more than 60 bits but hopefully less than 80 bits (the *Prefix* field can be less than 20 bits). Consequently, our experiments are fair enough to the CBF-based method. Even with such setting, the trace-driven evaluations demonstrate that both the redesigned Trie and FT achieve near-accurate multiset reconciliation, with 4.31 and 2.96 times faster on average than the CBF-based method, respectively. By contrast, to improve the reconciliation accuracy, CBF has to initialize more counters to represent the multisets.

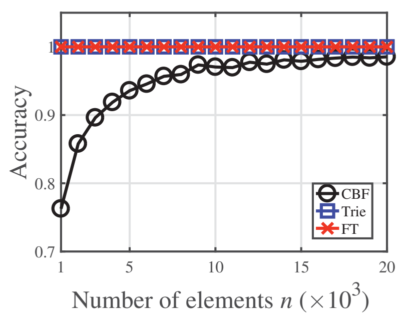
5.3 Performance evaluation with synthetic datasets

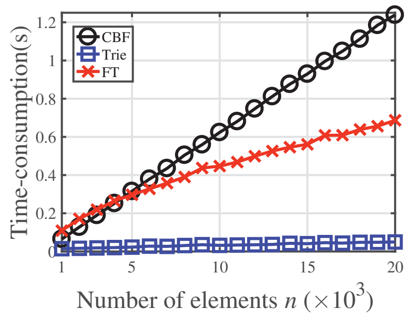
In this section, we further quantify the impact of parameters towards our proposals. We mainly consider the number of elements n, the number of different elements d and the ratio r between d_E and d.

5.3.1 *The impact of varied set cardinality* n

We first evaluate the impact of n, i.e., the number of elements in each multiset, on the reconciliation performance. Given $d{=}800$ and $r{=}0.5$, the reconciliation accuracy and time-consumption when n increases from 1,000 to 20,000 are depicted in Fig. 7(a) and Fig. 7(b), respectively. Note that, for the CBF-based method, the size of CBF table m is set as $20{\cdot}n$ and the number of hash functions is fixed as $k{=}3$. Apparently, except for the CBF-based method, other

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering


(a) Accuracy.


(b) Time-consumption.

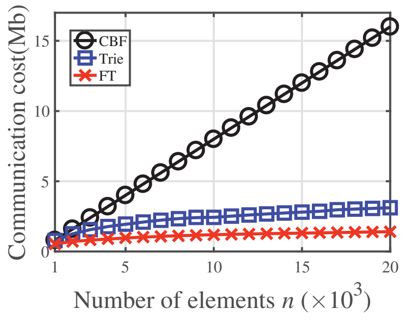
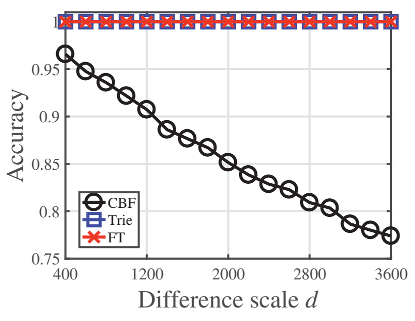
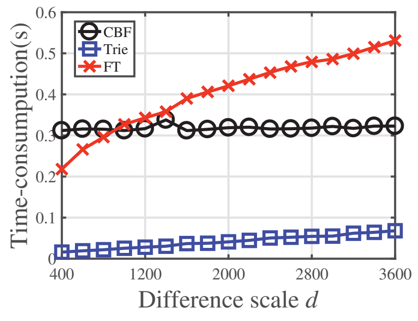

(c) Communication overhead.

Fig. 7. The impact of varied n on the performance of our proposals, where d=800, r=0.5.


(a) Accuracy.


(b) Time-consumption.

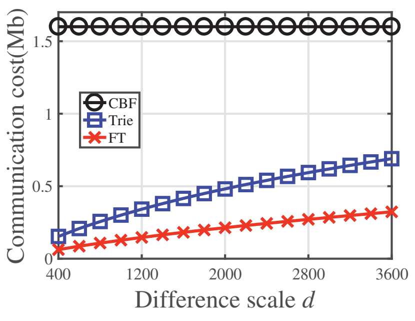

(c) Communication overhead.

Fig. 8. The impact of varied d on the performance of the proposals, where n=5000, r=0.5.

reconciliation strategies realize near 100% accuracy. As depicted in Fig. 7(a), the accuracy of the CBF-based method remains growing with the increase of n. The tendency of the CBF-based method can be explained with literature [27], which realize simple set reconciliation with CBF. For the 800 different elements, larger n means larger m, thus the probability that any of different element involves hash conflicts gets lower. Additionally, the CBF-based method has lower accuracy when dealing with multiset reconciliation than the simple set reconciliation scenario. This is because multiset reconciliation needs to derive what the different elements are, as well as the associated multiplicities. After subtracting, the counters in CBF may report the different elements correctly but may return an inaccurate multiplicity.

The corresponding time-consumption of our methods are reported in Fig. 7(b). All these methods take more time to query, compare, prune, or decode the counters or nodes with the increase of n. Among these methods, Trie costs the least time, while the CBF-based method needs the most time. The time-consumption of the CBF-based method is linearly increased when n grows. The hosts query the local elements against the subtracting result CBF_C. Consequently, more elements means more queries. For the redesigned FT, the children of any Carry node are certain and the range of elements that it represents can be derived directly (shown in Section 2.1.4). Therefore, when traverse or prune, the children can be accessed directly. Again, the reconciliation process for a redesigned Trie is much faster since the $prefix$ field helps to identify most of the different internal nodes.

Moreover, we calculate the caused communication overhead when n increases from 1,000 to 20,000. The CBF-based method has to exchange the entire counter vector. In our setting, a CBF vector has $20n$ counters each of which consumes 4 bits. Therefore, the total communication overhead of exchanging CBFs is $80n$ bits. The Trie-based and FT-based methods transmit the nodes partially and therefore saving bandwidth. Indeed, with given d and increased n, the communication overhead of transmitting Trie and FT nodes grows gradually. The reason is that to accommodate more elements, both the redesigned Trie and FT have more levels of nodes. As a result, the 800 different elements will affect more upper-level nodes.

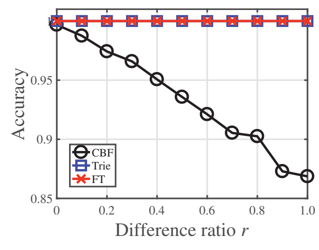
Generally, the value of n affects both the time-consumption and communication overhead of exchanging the data structures for all the proposals. The reconciliation accuracy of our Trie and FT based method is always near 100% when n varies. The CBF-based method, on the contrary, realize lower reconciliation accuracy than our proposals and its performance varies when n grows.

5.3.2 *The impact of varied difference scale* d

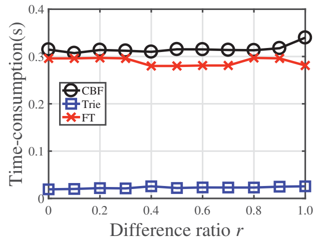
Similarly, we also evaluate the impact of d on the reconciliation performance with given r=0.5 and n=5,000. We increase d from 400 to 3,600 and record the resultant reconciliation accuracy, time-consumption, and communication overhead in Fig. 8. As depicted in Fig. 8(a), due to the intrinsic false positive and false negative errors of reconciliation, the CBF-based method fails to achieve accurate reconciliation. With more differences are introduced into the multisets, the accuracy of the CBF-based method decreases from 0.96 to 0.75, while the other methods always remain almost 100% accuracy. With the increase of d, the subtracting result CBF_C will contain more elements. As a consequence, more elements will suffer from the false positive or false negative errors. This intuitively explains why the CBF-

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering



(a) Accuracy.



(b) Time-consumption.

Fig. 9. The impact of varied r on the performance of the Trie-enabled method, where n=5000, d=800.

based method experiences the accuracy degradation when d grows.

Correspondingly, as depicted in Fig. 8(b), the CBF-based method takes constant time to accomplish the reconciliation process since its time-consumption is only determined by the value of n. However, all the other methods suffer from increasing time-consumption with the growth of d. For the Trie-based method, the traverse algorithm must compare more Trie nodes to identify these different elements for larger d. Naturally, larger d means more different Carry nodes in the subtracted FTs; hence, the FT-based method needs more time to distinguish these nodes. When $d \geqslant 1,000$, the redesigned FT costs more time than CBF. Still, Trie consumes the least time due to the *Prefix* field.

Fig. 8(c) further plots the communication overhead of related multiset reconciliation methods with n=$5,000$, when d ranges from 400 to 3,600. With each counter in CBF as 4 bits, and the size of the CBF table m=$20n$, the bpe (bits per element) of CBF is derived as 80. Consequently, the CBF-based method incurs constant communication overhead for each host, i.e., $1.6 \cdot 10^6$ bits. By contrast, for both the Trie-based and FT-based method, using the partial transmission strategy brings an incremental communication cost when d grows. Additionally, with the growth of d, the increasing trends of both the Trie-based and FT-based methods become smooth. The reason is that Trie and FT are typical tree-like data structures, and distinct leaf nodes may share common ancestors. An ancestor has to be transmitted, no matter how many of its children belongs to D. As a result, the value of d has a marginal impact on the communication overhead. Note that, with the growth of d, the incurred communication overhead for both the redesigned FT and Trie may reach their upper bound, i.e., all the nodes in the redesigned FT and Trie should be transmitted.

Generally, for the near-accurate reconciliation methods, their accuracy will be merely affected by the value of d, but their communication overheads, as well as the time-consumption, are proportional to d. For the CBF-based method, on the other hand, more different elements lead to lower reconciliation accuracy, stable time-consumption and constant communication overhead.

5.3.3 *The impact of diverse difference ratio* r

As aforementioned, the difference between two multisets consists of D_E and D_M. Thus in this section, we explore whether the ratio between d_E to d will bring significant influence to the reconciliation performance of our proposals. We set the value of n and d as 5,000 and 800, but vary r from 0 to 1. Fig. 9(a) and Fig. 9(b) report the corresponding reconciliation accuracy and time-consumption, respectively. Note that, the communication overhead of identifying the difference is determined by the values of n and d, but not related to r. Hence, we only quantify the impact on the reconciliation accuracy and time-consumption.

As shown in Fig. 9(a), the reconciliation accuracy of the CBF-based method exhibits a dramatic decrease along with the growth of r. In contrast, both the other methods keep near 100% reconciliation accuracy, in spite of the variation of r. In effect, larger r brings more diverse elements in the root sets A^* and B^*. As a result, the resulted CBF_C must accommodate more diverse elements, which causes a higher probability of false positive and false negative errors. We can see from Fig. 9(b) that the CBF-based method, the Trie-based method and the FT-based method experience stable time-consumption when r grows. The reason is that no matter the difference is caused by diverse multiplicities or elements, the corresponding nodes in the redesigned Trie and FT will be distinct. Therefore, we conclude that the value of r impacts the accuracy of the CBF-based method but will not affect the performance of the redesigned Trie and FT.

As a summary of the simulations, the redesigned Trie and FT always achieve near-accurate reconciliation in spite of the variations of parameters. The CBF-based method, however, requires much more space overhead to realize high reconciliation accuracy. The time-consumption of all the compared methods will be affected by n and d, but not r. The communication overhead of the three methods, increase significantly with the growth of n and d. With the above results, we believe our proposals outperform the CBF-based method in terms of reconciliation accuracy, time-consumption, and communication overhead at most time.

6 Discussion

In this section, we further discuss several related issues about the proposed methods.

The unsorted and sorted multisets. In this paper, we distinguish the unsorted multiset with sorted multiset and then redesign Trie and FT to achieve near-accurate reconciliations. One may hold that by hashing the unsorted multisets elements into a integer range and thereafter ranking these hash values, the unsorted multisets can be changed as sorted multisets. However, if we employ the FT-based method to reconciles the generated sorted multisets, extra elements may be misclassified as different elements. The reason is that FT recognize the different elements according to their locations in the Array nodes. Consequently, same elements in diverse locations will be treated as different ones.

The impact of hash collisions. Note that, we employ the hash functions to derive the *CHash* and *MHash* fields. An inherent characteristic of hash functions is hash collision where two or more distinct inputs generate a common hash value. In practice, with the given length of both *CHash* and *MHash* as 30 bits, the probability of hash collision is negligible. In each field, the probability of a hash collision is $1/2^{30}$. Correspondingly, the probability that both the *CHash* and *MHash* fields are collision-free will be $(1-1/2^{30})^2 \approx 1$. We declare that our methods are near accurate since they

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

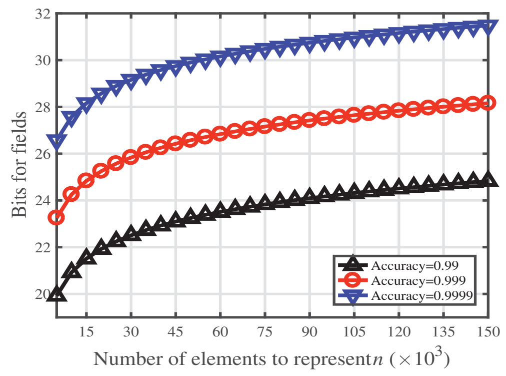


Fig. 10. The required bits for each field in an internal node and Carry node, with given reconciliation accuracy.

may miss some different elements due to the potential hash collisions.

Bits in each field. Consider that the accuracy of our method is $(1-1/2^b)^{2n}$, we can inversely derive how many bits should be allocated to each field of the nodes in Trie and FT, with respect of the given reconciliation accuracy. As shown in Fig. 10, with given accuracy, the number of bits for each field logarithmically increases to represent more elements. Moreover, to enable more accurate reconciliation, Trie and FT require more bits for each field to reduce the probability of hash collision. Therefore, in real use, the number of bits for each field in both Trie and FT can be adjusted to realize the accuracy guarantee.

The construction and maintenance of Tries and FTs. Inserting an element into a Trie costs $O(\mathcal{I})$ time-complexity, where $\mathcal{I}$ is the length of the identifiers. To represent a multiset with n elements, the total time-complexity will be $O(\mathcal{I}\cdot n)$. The value of $\mathcal{I}$ is usually a given constant, thus the time-complexity can be simplified as $O(n)$. By contrast, both inserting and updating an element in FT cost $O(\log n)$ time-complexity. Both Trie and FT support dynamic update of the recorded elements, which makes them practical in real systems. Despite the support of element deletion, CBF will incur increasing false positive and false negative errors during reconciliation when n grows. Once the Trie or FT in a host has been initialized, the later update only needs to insert or delete related elements, rather than reconstruct the whole Trie or FT. The CBF, however, calls for reconstruction when n exceeds its capacity.

Partial transmission strategy. Indeed, based on our partial transmission strategy, there is a trade-off between the communication overhead and the delay of subtracting process. On one hand, a fine-grained transmission (transmitting the nodes one by one in an extreme case) generates the least communication overhead since almost all the same nodes will be recognized and pruned. However, this calls for the most rounds of communication sessions, thereby leading to the longest delay. On the other hand, a coarse-grained transmission (e.g., exchanging all the nodes together) results in high communication overhead, with the least delay. An eclectic solution is to transmit the nodes in a batched manner. For instance, we can transmit 10 nodes together in each round of communication, such that the total rounds of communications will be decreased significantly and the pruning process can be completed with acceptable delay. Besides, if the user can evaluate the value of d in advance, the internal nodes with high probability to cover the elements in D will be transmitted together with one communication session. By contrast, the internal nodes may not aggregate elements in D with high probability will be sent partially. As a result, the delay will be further controlled.

7 Conclusion

In this paper, we motivate to characterize and tackle the reconciliation problem of two multisets, which is an essential task for various distributed applications. We argue that the methods based on probabilistic data structures fail to achieve satisfying reconciliation accuracy, due to the unavoidable false positives and false negatives. Accordingly, we redesign the Trie and FT data structure to represent unsorted and sorted multisets, respectively. Besides, to further reduce the communication overhead, we design a partial transmission strategy for the redesigned Trie and FT. The trace-driven evaluations demonstrate that Trie and FT achieve near-accurate multiset reconciliation (at the level of $1-10^{-4}$), with 4.31 and 2.96 times faster than the CBF-based method, respectively. The simulations based on synthetic datasets further indicate that our proposals outperform the CBF-based method irrespective of the parameter setting at most time.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," *in IEEE MSST,* Incline Villiage, NV, USA, 2010.

[2] J. Liu, S. Ahmad, E. Buyukkaya, R. Hamzaoui, and G. Simon,"Resource allocation in under-provisioned multi-overlay peer-to-peer live video sharing services," *Peer-to-Peer Networking and Applications,* pp. 1–15, 2014.

[3] S. Guo, Y. Gu, B. Jiang and T. He, "Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links," *IEEE Transactions on Computers,* vol. 63, no. 11, pp. 2787–2802, 2014.

[4] P. Kyasanur, R. R. Choudhury, and I. Gupta, "Smart gossip: An adaptive gossip-based broadcasting service for sensor networks," *in IEEE MASS,* Vancouver, BC, Canada, 2006.

[5] K. P. N. Puttaswamy, C. C. Marshall, V. Ramasubramanian, P. Studei, D. B. Terry, and T. Wobber, "Docx2go: Collaborative editing of fidelity reduced documents on mobile devices," *in ACM MobiSys*, San Francisco, CA, USA, 2010.

[6] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference?: Efficient set reconciliation without prior context," *in ACM SIGCOMM,* Toronto, Ontario, Canada, 2011.

[7] L. Luo, D. Guo, J. Wu, O. Rottenstreich, Q. He, Y. Qin, and X. Luo, "Efficient multiset reconciliation," *IEEE/ACM Transactions on Networking,* vol. 285, no. 2, pp. 1190-1205, 2017.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM,* vol. 13, no. 7, pp. 422–426, 1970.

[9] L. Fan, P. Cao, J. Almeida, and A. Broder. "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking,* vol. 8, no. 3, pp. 281-293, 2000.

[10] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, "Robust set reconciliation," *in ACM SIGMOD,* Snowbird, Utah, USA, 2014.

[11] G. Karpovsky, B. 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.

[12] K. Abdel-Ghaffar, and A. Abbadi, "An optimal strategy for comparing file copies," *IEEE Transactions on Parallel and Distributed Systems,* vol. 5, no. 1, pp. 87-93, 1994.

[13] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory,* vol. 49, no. 9, pp. 2213-2218, 2004.

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2849997, IEEE Transactions on Knowledge and Data Engineering

[14] N. Alon, and A. Orlitsky. "Source coding and graph entropies," *IEEE Transactions on Information Theory,* vol. 42, no. 5, pp. 1329-1339, 1996.

[15] D. Singh, A. Ibrahim, T. Yohanna, and J. Singh, "An overview of the applications of multisets", *Novi Sad Journal of Mathematics,* vol. 37, no. 3, pp. 73-92, 2007.

[16] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," *in ACM SIGCOMM,* Florianpolis, Brazil, 2016.

[17] D. L. B. Rene, "File searching using variable length keys" *in Western Joint Computer Conference,* San Francisco, CA, USA, 1959.

[18] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Software Practice and Experience,* vol. 24, no. 3, pp. 327-336, 1996.

[19] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, pp. 490-500, 1960.

[20] A. Andersson, and S. Nilsson, "Improved behaviour of tries by adaptive branching," *Information Processing Letters,* vol. 46, no. 6, pp. 295-300, 1993.

[21] B. Rais, P. Jacquet, and W. Szpankowski. "Limiting distribution for the depth in Patricia tries," *SIAM Journal on Discrete Mathematics,* vol. 6, no. 2, pp. 197-213, 1993.

[22] P. Mishra, "A New algorithm for updating and querying sub-arrays of multidimensional arrays," *arXiv preprint,* arXiv:1311.6093, 2013.

[23] J. Vuillemin, "A data structure for manipulating priority queues," *Communications of the ACM,* vol. 21, no. 21, pp. 309-315, 1978.

[24] A.V. Aho, and J.E. Hopcroft, "The design and analysis of computer algorithms," *Pearson Education India,* 1974.

[25] C.A. Crane, "Linear lists and priority queues as balanced binary trees," *Department of Computer Science, Stanford University,* Stanford, CA, USA, 1972.

[26] CAIDA UCSD Anonymized Internet Traces 2016-equinix-chicago.dirB.20160406-140200.UTC, http://www.caida.org/data/passive/passive_2016_dataset.xml.

[27] 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.

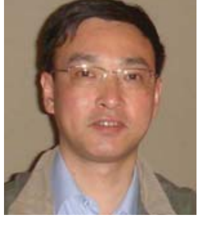
Lailong Luo received his B.S. and M.S. degree at the College of Systems Engineering from National University of Defence Technology, Changsha, China, in 2013 and 2015, respectively. He is currently working toward a Ph.D degree in College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include data structure and distributed networking systems.

Deke Guo received his B.S. degree in Industry Engineering from Beijing University of Aeronautic and Astronautic, Beijing, China, in 2001, and the Ph.D. degree in Management Science and Engineering from National University of Defense Technology, Changsha, China, in 2008. He is a Professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks.

Xiang Zhao received the Ph.D. degree from The University of New South Wales, Australia, in 2014. He is currently an assistant professor with the College of Systems Engineering, National University of Defense Technology, Changsha, China. His work has been published on various competitive venues, which include SIGMOD, VLDB, ICDE, ICDM, CIKM and VLDB Journal. His research interests include graph data management and mining.

Jie Wu is the chair of and a professor in the Department of Computer and Information Sciences, Temple University. Prior to joining Temple University, he was a program director at the US National Science Foundation. His research interests include wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has published more than 450 papers in various journals and conference proceedings. He has served as an IEEE Computer Society distinguished visitor and is the chairman of the IEEE Technical Committee on Distributed Processing (TCDP). He is an IEEE Fellow.

Ori Rottenstreich is the chief research officer of Orbs, Israel. He received the B.S. in Computer Engineering (summa cum laude) and Ph.D. degree from the Electrical Engineering department of the Technion, Haifa, Israel in 2008 and 2014, respectively. From 2015 to 2017, he was a Post-Doctoral Research Fellow at the Department of Computer Science, Princeton University. He was a recipient of the Rothschild Yad-Hanadiv Post-Doctoral Fellowship and the Google Europe Ph.D Fellowship in computer networking. He is mainly interested in computer networks and especially in exploring novel coding techniques for networking applications.



Xueshan Luo received his B.E. degree in Information Engineering from Huazhong Institute of Technology, Wuhan, China, in 1985, and his M.S. and Ph.D degrees in System Engineering from the National University of Defense Technology, Changsha, China, in 1988 and 1992, respectively. Currently, he is a professor of College of Systems Engineering, National University of Defense Technology. His research interests are in the general areas of information system and operation research.

1041-4347 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.