MCFsyn: A Multi-party Set Reconciliation Protocol with the Marked Cuckoo Filter

Lailong Luo, Deke Guo, Yawei Zhao, Ori Rottenstreich, Richard T.B Ma, and Xueshan Luo

***Abstract*—Multi-party set reconciliation is a key component in distributed and networking systems. It naturally contains two dimensions, i.e., set representation and reconciliation protocol. However, existing sketch data structures are insufficient to satisfy the new needs brought by the multi-party scenario simultaneously, including space-efficiency, mergeability, and completeness. The current reconciliation protocols, on the other hand, fail to achieve the global optimization of communication cost. To this end, in this paper, we propose the marked cuckoo filter (MCF), a data structure for representing set members. Grounded on MCF, we implement the MCFsyn protocol to reconcile multiple sets. MCFsyn aggregates and distributes sets information represented by MCFs along with an underlying minimum spanning tree among the participants. The participants then identify the different elements by traversing the overall MCF which contains the information of all elements in the union set. For the identified missing elements, MCFsyn helps the participants to choose the optimal senders to fetch with the minimum communication cost. Comprehensive evaluations indicate that MCFsyn significantly outperforms existing alternatives in terms of both reconciliation accuracy and communication cost.**

***Index Terms*—Set reconciliation, minimum spanning tree, marked cuckoo filter, accuracy, communication cost**

I. Introduction

As users migrate their computation and data to the cloud, cloud-based services such as Dropbox, Google Drive, and OneDrive have emerged to enable users to access data from various devices and allow team collaborations over the same data. Since multiple copies of the data exist in users' devices, cloud servers, and edge servers, as users update them possibly from different devices simultaneously, these multiple copies need to be periodically reconciled or synchronized for their consistency and correctness. This so-called *multi-party set reconciliation* problem also occurs in wireless sensor networks [1], software-defined networks [2], content delivery networks [3], blockchain transaction pools [4] and beyond.

The multi-party set reconciliation problem can be naturally decomposed and tackled from two dimensions: 1) set representation — how the set elements are represented; and 2) reconciliation protocol — how the participants interact with each other to identify and thereafter transfer the different elements. Existing set reconciliation methods mainly rely on linear sketch data structures (e.g., hash tree, Bloom filter (BF) [5] [6], Invertible Bloom filter [7], Invertible Bloom lookup table (IBLT) [8] [9], Invertible Counting Bloom filter (ICBF) [10], etc) to represent set elements. In particular, if set elements can be represented as integers, the characteristic polynomial can also work as a sketch of the corresponding set [11] [12]. The reconciliation protocol is thereafter built on top of these set sketches. Usually, these sketches are exchanged among reconciliation participants. With the sketches from others, a local participant can identify (with high accuracy) the particular elements which have to be transferred for reconciliation.

However, existing sketch data structures are insufficient to satisfy the new needs brought by the multi-party set reconciliation scenario. Specifically, a sketch data structure in the multi-party context should be: 1) **space-efficient**, the used space should be much less than the original data size; 2) **mergeable**, multiple such data structures can be merged as one without loss of information; and 3) **complete**, both the content information (such as a fingerprint representing element identity) and the affiliation information (the sets an element belongs to) are correctly represented. The space-efficiency and mergeability features guarantee the low communication cost of exchanging the sketches among the reconciliation participants. The completeness property further ensures reconciliation accuracy. On the contrary, existing sketch data structures fail to achieve them simultaneously. Bloom filter and its variants are space-efficient, but most of them are often not mergeable. The IBLT is both space-efficient and mergeable, yet fails to achieve the completeness property.

Moreover, existing reconciliation protocols do not optimize the communication cost with the joint consideration of the positions of participants in the network and the data distribution among the participants. Accordingly, they usually result in redundant sketch exchange and data transfer. Currently, most cloud storage services implement the backup strategy to reconcile sets. They usually maintain the cloud server as the central party to gather all the elements from the participants. Participants upload their modified data or download their missing data from the central party. Unfortunately, the overall cost of this strategy highly depends on the physical locations of participants and the distance between them. It may waste network bandwidth since a participant can only get its missing elements from the central party, even though its neighbor may also hold that element. Other possible protocols, such as exchanging the sketches and different elements through the

Lailong Luo, Deke Guo, and Xueshan Luo, are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan, 410073, China. E-mail: {luolailong09, dekeguo, xsluo}@nudt.edu.cn.

Yawei Zhao is with China Electronic Equipment System Engineering Company, Minzhuang Road 77, Haidian District, Beijing, 100039, China. E-mail: csyawei.zhao@gmail.com.

Richard T. B. Ma is with the School of Computing, National University of Singapore, Singapore. E-mail: tbma@comp.nus.edu.sg.

Ori Rottenstreich is with the Israel Institute of Technology and ORBS Research, Israel. E-mail: or@cs.technion.ac.il.

Lailong Luo is the corresponding author.

all-to-all transmission or gossip protocol, occupy the network bandwidth in an even more aggressive manner.

For the above reasons, in this paper, we first propose a new variant of cuckoo filter [13], named the *marked cuckoo filter* (MCF), to represent the sets in each reconciliation participant. The MCF attaches an additional field in each slot to describe which set(s) the stored fingerprint belongs to (earlier to the reconciliation). This field is called the *mark* field. For example, given three sets S_1, S_2 and S_3, it has three bits in each MCF slot (in addition to the fingerprint) to indicate the affiliation of the stored element. The i^{th} bit will be set to 1 if S_i contains that element. MCF naturally inherits the functionalities from the standard cuckoo filter, including element insertion, query, and deletion. Additionally, MCF enables filter-level operations, such as aggregation and subtraction, so that the MCFs from different sets can be aggregated together or subtracted pairwisely. Such a design allows MCF to be space-efficient, mergeable, and complete.

Based on MCF, we propose a novel multi-party set reconciliation protocol named MCFsyn. MCFsyn has the following five main steps : (i) each participant represents its set elements as an MCF vector; (ii) MCFsyn aggregates the MCF sketches from all sets as an overall MCF in the central relay participant; (iii) the central relay participant distributes the overall MCF to all other participants; (iv) the participants determine the missing elements by traversing the overall MCF; and (v) the participants try to pull these missing elements with the least transmission cost. Each of the above steps needs a careful design with the joint consideration of reconciliation delay, communication cost, and network topology. The main contribution of this paper can be summarized as follows:

- We propose the MCF data structure to represent set elements. Besides the fingerprint field, MCF uses an additional mark field in each slot to record the affiliation of the stored element. The space-efficiency, mergeability, and completeness properties make MCF an elegant sketch in the multi-party set reconciliation scenarios.
- We design the MCFsyn protocol to reconcile sets of the multiple participants. MCFsyn relies on an underlying minimum spanning tree of the participants to aggregate and distribute the MCFs. Then a participant P_i tries to pull any missing element x with the minimum communication cost from a preferred participant which hosts x.
- Comprehensive evaluations are conducted to quantify the performance of MCFsyn. Specifically, in our evaluations, MCFsyn generates 20x and 70x times fewer errors on average than the methods enabled by BF and IBLT with the same bits per element, respectively. Moreover, MCFsyn causes 21x and 7.9x times less communication cost on average than transferring BF with the all-to-all scheme and exchanging IBLT with the gossip protocol, respectively.

The rest of this paper is organized as follows. Section II introduces the background and related work. Section III presents a novel Cuckoo filter variant named marked Cuckoo filter. Section IV details the design philosophy of our MCFsyn protocol. Section V presents the theoretical performance analysis for MCFsyn. Section VI reports the evaluation results and at last Section VIII concludes the whole paper.

II. Background and Related Work

A. Set Reconciliation

The input to the multi-party set reconciliation are n hosts (or parties), each of which has a set $S_i \subseteq U$ (where U is the universal from which elements are taken). The target is to identify and thereafter exchange the different elements among these sets, so that after reconciling $S'_1 = S'_2 = \cdots = S'_n = S = \cup_i S_i$. One previous approach for two-party set reconciliation uses characteristic polynomials [11] [12] coupled with coding theories such as Reed-Solomon codes, BHC codes, etc. This kind of methods treat each element as an integer value. For set S_1, its characteristic polynomial is calculated as:

$$\chi_{S_1}(Z) = \prod_{x_i \in S_1} (Z - x_i), \tag{1}$$

The set S_2 also derives out its $\chi_{S_2}(Z)$. Thereafter, in the following rational function

$$\frac{\chi_{S_1}(Z)}{\chi_{S_2}(Z)}, \tag{2}$$

the common elements are eliminated. The sum of the degrees of the numerator and denominator is at most d, where $d = |S_1 \backslash S_2| + |S_2 \backslash S_1|$. Interpolation is then executed to determine the integer values of different elements demonstrated in the above rational function. However, this division and interpolation method is somewhat time-consuming ($O(d^3)$ time-complexity) and it is computation-intensive to recover all elements using Gaussian elimination.

Recent set reconciliation methods rely on randomized data structures to provide a sketch of sets. After exchanging and comparing these sketches, the different elements can be determined and transferred reasonably. These data structures include Merkle tree [14] [15], Counting Bloom filter (CBF) [5], Invertible Bloom filter (IBF) [7], IBLT [8] [9], Invertiable Counting Bloom filter (ICBF) [10], etc. The nodes in the Merkle trees are compared to prune same sub-trees, thereby deriving out the different elements between two sets. CBF, IBF, IBLT, ICBF, on the contrary, subtract the filters to eliminate the existence information of common elements. The remained different elements are decoded from the subtraction results through either query or listing mechanisms.

As for multi-party set reconciliation, it is usually decomposed as multiple rounds of two-party set reconciliation with the above methods. We note that most-recently, the IBLT and characteristic polynomial are generalized from two-party set reconciliation to multi-party scenarios [16] [17]. Specifically, the binary fields in IBLT are extended to multi-ary to aggregate element information [16]. The XOR operations in the IBLT are also redefined in the multi-ary system correspondingly. The possibility of using characteristic polynomials to reconcile multiple sets is investigated by combining with the gossip protocol in general networks [17].

Another related model is the multi-party membership query, which answers the question which set(s) an element belongs to. The existing solutions for this query are mainly based on

Bloom filter variants [18] [19] [20] [21] [22]. The Combinatorial Bloom filter indicates the affiliation information of any element with different groups of hash functions [18]. The OMASS proposes to isolate the values generated from hash functions in a sub-block for diverse sets with differentiated hash functions [19]. In Noisy Bloom filter, an element x is mapped to k bits in the bit vector. The bits after these k bits are employed to explicitly record the affiliation information of x [20]. The Difference Bloom filter makes the representation of elements exclusive by writing a different number of 1s and 0s in the same filter [21]. Particularly, the coloring embedder first maps elements to a high dimensional space to almost eliminate hashing collisions and then uses a dimensional reduction representation to save memory [22]. However, these methods assume that an arbitrary element x exclusively belongs to a single set, making these proposals impractical in our cases.

B. Cuckoo Filter and its Variants

Cuckoo filter (CF) [13] is a light-weight probabilistic data structure to support constant-time membership query. An element x is associated with a f-bit fingerprint η_x which is derived out by a hash function h_0. Unlike Bloom filter, CF stores the fingerprint of each element directly. Structurally, a CF consists of m buckets, each of which is capable of residing b fingerprints. An CF offers 2 candidate buckets to each element, and the fingerprint can be stored in either of the candidates. If both candidates are occupied, CF randomly kicks out an existing fingerprint in one of the candidates and reinserts the victim in its alternative candidate bucket. This re-allocation ends successfully when a bucket has available space or fails when the number of such re-allocations reaches a given threshold *max*. During reallocation, the alternative bucket can be derived out by executing an XOR operation towards the current bucket and the fingerprint of the victim. Specifically, the two candidate buckets are derived as $h_1(x){=}hash(x)$ and pair-wisely $h_2(x){=}h_1(x) \oplus hash(\eta_x)$.

Most recently, several CF variants have been proposed to further improve its performance [23] [24] [25] [26]. The Simplified Cuckoo filter [23] (SCF) calculates the indices of buckets for an element x as $h_1(x)$ and $h_1(x){\oplus}\eta_x$. In this way, SCF provides a theoretical analysis of its performance with the graph theory. The Adaptive cuckoo filter [24] tries to remove false positive errors from the CF vector by resetting the collided fingerprints with optional hash functions. Dynamic Cuckoo filter [25] (DCF) adaptively maintains multiple homogeneous CFs with same parameter setting to enable the filter-level capacity elasticity. Consistent Cuckoo filter (CCF) further realizes the bucket-level capacity elasticity by introducing a consistent hash ring into each CF [26]. The buckets in each CF are mapped onto the consistent hash ring so that the buckets can be added or removed at will. Thereafter, the element fingerprints are also mapped onto the consistent hash ring to determine their candidate buckets on the ring. The upper bounds of false positive rate in both DCF and CCF are $1-\prod_{i=1}^{s}(1-\xi_i)$, where s is the number CFs in DCF and CCF and ξ_i is the false positive rate of the i^{th} CF. There are also some lock-free designs for cuckoo hashing [27]. Besides, Morton filter speeds up the insertion, query and deletion operations by organizing the buckets as compressed blocks and pruning unnecessary memory accesses [28]. Vacuum filter [29] achieves a significant improvement of throughput by confining the two candidate buckets for any element in an *alternate range* which can be fetched by exactly one memory access.

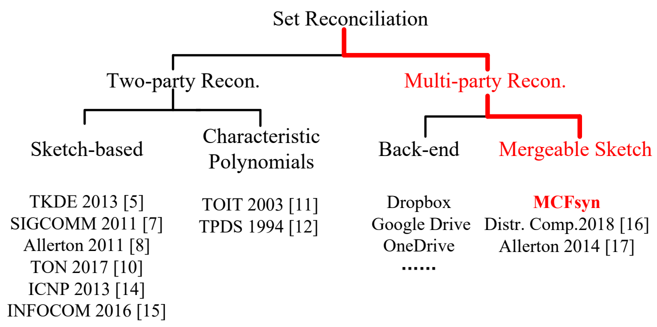


Fig. 1. The existing work for set reconciliation. MCFsyn differs from others by implementing mergeable sketches and optimizing the communication cost with a customized reconciliation protocol.

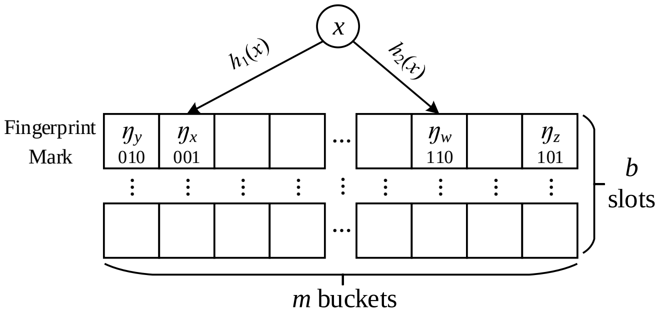


Fig. 2. A toy example of Marked Cuckoo Filter (MCF) which represents three sets, including $S_1{=}\{x,z\}$, $S_2{=}\{y,w\}$, and $S_3{=}\{w\}$. The mark field has three bits to explicitly indicate whether the represented element is a member of S_1, S_2, and S_3 (from right to left), respectively. For instance, the element x that belongs only to S_1 is associated with the mark 001 and w that belongs to both S_2,S_3 with 110.

Generally, the above CF variants are space-efficient and mergeable, however, fail to achieve the completeness property (they cannot represent the affiliation information of elements). Therefore, we investigate the MCF to represent multi-party set members. As shown in Fig. 1, our MCFsyn dedicates to solve the multi-party reconciliation problem in distributed systems, it implements the mergeable sketches to aggregate and distribute set information and tries to minimize the overall communication cost.

III. The Marked Cuckoo Filter

In this section, we present the MCF design, including its data structure and operations to represent multi-party sets.

A. The MCF Data Structure

Besides the fingerprint information of elements, multi-set representation also requires the affiliates of the stored elements. Therefore, MCF attaches additional bits in each slot to integrate element affiliation information.

As depicted in Fig. 2, MCF consists of m buckets, each of which has b slots to accommodate at most b element fingerprints. Each slot has two fields, including the fingerprint field to represent element fingerprint information and the mark field showing the affiliation information of the element. The fingerprint field has f bits to store the element fingerprint; while the mark field occupies n bits (correspond to the number of participants n) to explicitly label the affiliation(s) of the stored fingerprint. The i^{th} bit in the mark field will be set to 1 if the represented element in this slot is a member of the set S_i. Just as CF, MCF offers two candidate buckets for each element to represent with hash functions $h_1(x)$=$hash(x)\%m$ and $h_2(x)$=$h_1(x)\oplus(hash(\eta_x)\%m)$, respectively. The fingerprint η_x can be stored in either of its candidate buckets. Upon insertion, if both candidate buckets are fully occupied, MCF randomly kicks out an existing fingerprint from one of the candidate buckets to store η_x. The victim will be reallocated to its alternative candidate bucket. The re-allocation ends successfully when a bucket has available space or fails when the number of re-allocations reaches a given threshold *max*.

The MCF naturally supports multi-party set representation by jointly considering the fingerprint and mark field in slots. Therefore, it acts as the basis of our multi-party set reconciliation protocol.

B. MCF Operations

With the above design, MCF provides element-oriented operations to represent sets, including element insertion, query, and deletion. Besides, some inter-filter operations such as aggregation and subtraction are also enabled.

Element insertion. To insert an arbitrary element x in set S_i, we first generate its f-bit fingerprint with the hash function $h_0(x)$. The candidate buckets for x are then calculated by the functions $h_1(x)$ and $h_2(x)$. If either of its candidate buckets has an empty slot to accommodate x, the fingerprint η_x will be stored there and the i^{th} bit of the mark field will be set to 1. Otherwise, MCF has to kick out a stored fingerprint randomly and clear its mark field from the two candidate buckets to represent x. The victim will be re-allocated to its alternative candidate bucket. The re-allocation will end until there is no further victim or the number of re-allocations reaches the given upper bound *max*. Notice that when a victim is kicked out, the 1s in mark field of that slot will be reset as 0s. Pair-wisely, when the victim is reinserted, the corresponding mark field bit(s) in the target slot will be set to 1s. This guarantees the correctness of the recorded affiliation information.

Element query. To query an arbitrary element x with fingerprint η_x, MCF just checks the two candidate buckets. If the fingerprint can be found in either of its candidate buckets, MCF returns the mark field to explicitly demonstrate the set(s) x belongs to. Otherwise, MCF returns *False* to indicate that x is not a member of any set. The time-complexity of query is constant since only two candidate buckets are checked. Similar to the standard cuckoo filter, the false positive rate of a query (wrongly indicating a membership is some set the element does not belong to) is bounded as $\xi \leq 1-(1-\frac{1}{2f})^{2b}$, where f and b are the number of bits in each fingerprint and the number of slots in each bucket, respectively.

Algorithm 1: MCF Aggregation

```
Input: Two MCFs to aggregate: $MCF_i$ and $MCF_j$
Output: The aggregation result $MCF_o$
for $k$=0 to $m-1$ do
    for $r$=0 to $b-1$ do
        if $MCF_j[k][r]$.fingerprint$\neq \emptyset$ then
            Locate the $MCF_j[k][r]$.fingerprint in $MCF_i$,
              denoted as $slot$;
            if slot is not Null then
                $slot$.mark = $slot$.mark OR $MCF_j[k][r]$.mark;
            else
                Insert $MCF_j[k][r]$.fingerprint into $MCF_i$;
Let $MCF_o = MCF_i$;
return $MCF_o$
```

Algorithm 2: MCF Subtraction

```
Input: The two MCFs to subtract $MCF_i$ and $MCF_j$
Output: The $MCF_i$ and $MCF_j$ after subtracting
for $k$=0 to $m-1$ do
    for $r$=0 to $b-1$ do
        if $MCF_j[k][r]$.fingerprint$\neq \emptyset$ then
            Locate the $MCF_j[k][r]$.fingerprint in $MCF_i$,
              denoted as $slot$;
            if slot is not Null then
                Reset the common 1s in $slot$.mark and
                  $MCF_j[k][r]$.mark to 0s;
                Empty $slot$ and $MCF_j[k][r]$ when no 1s left
                  in their mark fields;
return $MCF_i$ and $MCF_j$;
```

Element deletion. Deletion is required for dynamic set representation. MCF supports both the deletion of element x from a specific set S_i and the elimination of element x from all its affiliates with only one execution. For example, to delete x from set S_i, MCF first locates the fingerprint η_x in the candidate buckets. If η_x cannot be found in its candidate buckets or the i^{th} bit in the mark field is 0, MCF returns *False* to declare that x is not a member of set S_i and the deletion has failed. Otherwise, MCF just resets the i^{th} bit in the mark field to 0 to indicate that x is not a member of S_i anymore. After that, if there are no 1s in that mark field, the fingerprint field will also be cleared. To eliminate x from all the sets, MCF simply tries to remove η_x and reset all the 1s in the mark field. If succeed, MCF returns *True*; otherwise, it returns *False*. The time-complexity of deleting an element is also constant.

MCF aggregation. Aggregation means to merge the fingerprint information and affiliation information from multiple homogeneous MCFs into one single MCF. This operation is important to reduce communication overhead during reconciliation. As shown in Algorithm 1, given MCF_i and MCF_j, the basic insight is to add the affiliation information of common fingerprints into MCF_i and insert sole fingerprint of MCF_j into MCF_i. Specifically, the algorithm needs to traverse the whole MCF_j vector. For any $MCF_j[k][r]$.fingerprint, we try to locate it in MCF_i. For MCF_i and MCF_j with the same parameter setting and hash functions, this is straightforward since they offer same candidate buckets for a common fingerprint. If this fingerprint can be found in MCF_i, the slot in MCF_i will *OR* itself with the mark field of the corresponding slot in MCF_j to aggregate the affiliation information (line 4 to 6). Otherwise, $MCF_j[k][r]$.fingerprint is inserted into MCF_i. Eventually, the updated MCF_i is returned as the aggregation result. The time-complexity of aggregation is $O(mb)$.

MCF subtraction. Given two homogeneous MCFs, e.g., MCF_i and MCF_j, subtraction tries to eliminate common fingerprints in them. Only the fingerprints appear in one single set or common fingerprints with diverse affiliations will be remained. To this end, as shown in Algorithm 2, we also have to traverse the whole MCF_j vector. For any $MCF_j[k][r]$.fingerprint, Algorithm 2 locates this fingerprint in MCF_i first. If this fingerprint can be found in *slot* of MCF_i, the common bits in the mark field of *slot* and $MCF_j[k][r]$ will be reset to 0s. In this way, the common affiliations of this fingerprint are removed. After such resetting, if there is no 1 in *slot*.mark or $MCF_j[k][r]$.mark, Algorithm 2 will empty the fingerprint fields in these two slots. On the other hand, if this fingerprint cannot be found in MCF_i, it will be remained in MCF_j. The time-complexity of subtraction is also $O(mb)$. Such subtraction functionality is useful for two-party set reconciliation and difference estimation.

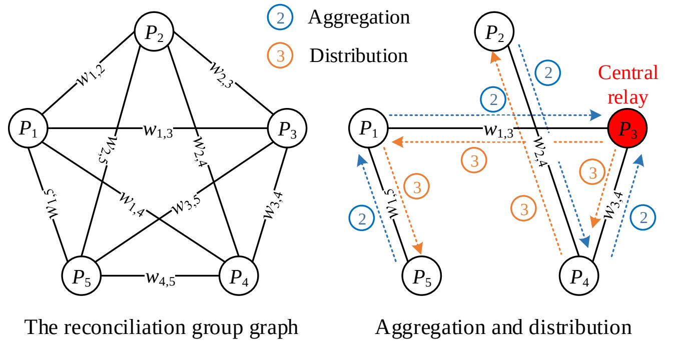
IV. The MCFsyn Reconciliation Protocol

A. The MCFsyn Framework

Multi-party set reconciliation means to deduce the different elements between sets and exchange these different elements such that every participant has the same set elements as the union of these sets. We provide a general framework for multi-party set reconciliation with the following five steps.

- Step 1: **Representation.** Every participant represents its local elements with the employed data structure (MCF in our proposal) to provide a sketch of the local set.
- Step 2:**Aggregation.** The sketches from all the participants are sent to a logical central relay wherein they are aggregated together to produce a sketch of the global union, $S = \cup_i S_i$. For privacy preservation, the data should only be stored and transferred within the group. Therefore, the logical central relay must also be a reconciliation participant.
- Step 3: **Distribution.** The logical central relay distributes the aggregation results to all the participants. By doing so, every participant acknowledges the union set.
- Step 4: **Extraction.** Each participant traverses the global sketch to determine its missing elements (elements which do not belong to the local set), as well as exclusive elements (elements that belong only to the local set).
- Step 5: **Transmission.** The different elements in $\cup_i S_i - \cap_i S_i$ are transmitted among the participants with selected senders to accomplish the reconciliation task.

The above framework is general while we specify it with our MCF. The reason is that the MCF data structure naturally provides the following features: 1) space-efficiency, the caused communication overhead in the aggregation and distribution steps is bounded and acceptable; 2) mergeability, a participant can aggregate the received MCFs with its local MCF as a single MCF before sending them out; 3) completeness, the mark field in MCF is responsible to record the affiliation of the stored elements. Additionally, MCFsyn can implement differentiated transfer strategies for the missing elements and exclusive elements to save the communication cost in the transmission step.



The reconciliation group graph

Aggregation and distribution

Fig. 3. Left: The reconciliation group graph. Right: The aggregation (dotted blue arrows) and distribution (dotted orange arrows) of sketches within the reconciliation group.

B. The Design Details of MCFsyn

With the above framework, there are still two challenges remained. First, the route of aggregation and distribution have significant impacts on the total communication overhead. Gossiping or broadcasting these sketches indeed works, however, leads to vast delay and communication cost. Therefore, reasonable routes are needed with the joint consideration of the underlying network. Second, the missing elements may be hold by multiple participants. It is challenging to select a provider so that the communication overhead is minimized. To settle these challenges, we first present the abstraction of the reconciliation group before elaborating our design details.

We consider n participants denoted as $P_1, \ldots, P_n$, each of which hosts a set of elements. As depicted in Fig. 3, we abstract the reconciliation group as a fully-connected graph. Each edge in the graph couples with a weight which measures the hops in the physical network between this pair of participants. For instance, the weight $w_{i,j}$ denotes the hops between participant P_i and P_j. In the aggregation step, all sketches are sent to the logical central relay. With the above abstraction, the design details are stated as follows.

Representation. Each participant represents its local set with an MCF. These MCFs are homogeneous with identical parameter settings, including filter length m, the hash functions for fingerprints and candidate buckets, and the number of slots in each bucket b. The homogeneity simplifies the subsequential aggregation and extraction steps significantly. These MCFs can also be heterogeneous with a slight alteration, as elaborated later in Section IV-C.

Aggregation. After formulating the reconciliation group graph, we derive out a minimum spanning tree (MST) of the graph as the route for the sketch aggregation. A single participant is regarded as the logic central relay to gather and distribute the MCFs from others. Note that the selection of the

Algorithm 3: MCFsyn Extraction at participant P_i

```
Input: The overall MCF $MCF_o$
Output: The missing elements and exclusive elements of $P_i$
1  Let $D_{M_i}$=$\emptyset$ denote the set of $P_i$'s missing elements;
2  Let $D_{E_i}$=$\emptyset$ denote the set of $P_i$'s exclusive elements;
3  for $k$=0 to $m-1$ do
4      for $r$=0 to $b-1$ do
5          if $MCF_o[k][r]$.fingerprint$\neq \emptyset$ then
6              if $MCF_o[k][r]$.mark[i]==0 then
7                  Add $MCF_o[k][r]$ into $D_{M_i}$;
8              else if Only $MCF_o[k][r]$.slot[i]==1 then
9                  Add $MCF_o[k][r]$.fingerprint into $D_{E_i}$;
10 return $D_{M_i}$ and $D_{E_i}$;
```

logic central relay has no impact on the communication cost of MCFsyn. We prefer to selecting the participant with the largest degree in the MST as the logic central relay. Such design enables the parallel transmission of sketches from the leaf participants to the central relay. In the example shown in Fig. 3, participant P_3 is selected as the central relay. Consequently, the two leaf participants P_5 and P_2 can send their MCFs along with the MST simultaneously. An internal participant aggregates the MCFs from its children (having longer distance to the central relay) with its local MCF and thereafter sends the aggregation results to its father participant. In the example depicted in Fig. 3, after receiving MCF_5 from participant P_5, participant P_1 aggregates its MCF_1 with MCF_5 and then sends the result to participant P_3. This aggregating-while-transmitting strategy reduces communication cost significantly.

Distribution. The central relay executes the aggregation operation provided by MCF, so that all the MCFs from its children are merged as an overall MCF, denoted as MCF_o. MCF_o represents all the elements in the union set $S = \cup_i S_i$ and their affiliation information. As shown in Fig. 3, MCFsyn distributes MCF_o from the central relay to others along with the generated MST.

Extraction. After receiving MCF_o, each participant tries to determine both the missing elements and exclusive elements by traversing MCF_o. As elaborated in Algorithm 3, participant P_i processes the received MCF_o. For any stored fingerprint in a slot, if the i^{th} bit in the corresponding mark field is 0, it means this fingerprint is not a member of S_i. Therefore, this fingerprint, together with its affiliation information, is added into D_{M_i} (line 6 to 7). On the other hand, if only the i^{th} bit in the corresponding mark field is 1, this element belongs to S_i solely and is added into D_{E_i} (line 8 to 9). The time-complexity of the extraction is $O(mb)$ since all the slots will be checked.

Transmission. Lastly, the participants exchange their different elements to finish the reconciliation task. For a participant P_i, its exclusive elements in D_{E_i} will be pushed to other participants by either broadcasting directly to save time or spreading along with the MST to achieve optimal communication cost. For a missing element in D_{M_i}, if its corresponding mark field only has one non-zero bit then this element is an exclusive element of another participant. Therefore participant P_i will do nothing but waiting for receiving that element content from that element holder. By contrast, if there are multiple 1s in that mark field, participant P_i selects the participant with which it establishes a link with the lowest weight as its provider. This strategy is implemented as a preference list based on the reconciliation group graph. In this list, a neighbour with a lower-weight link gets a higher preference. Consequently, the challenge of selecting the optimal element sender is resolved.

C. The Generalization of MCFsyn

We consider the generalization of MCFsyn by further investigating the using of heterogeneous MCFs and the scenario with dynamic reconciliation participants.

1) Heterogeneous MCFs: The above MCFsyn protocol implements homogeneous MCFs at each participants. However, in reality, a participant may hold much fewer elements than others due to long-term offline reasons or store much more elements than others because of a sudden content update. In this case, representing elements with homogeneous MCFs may not be economic. Consequently, we consider the using of heterogeneous MCFs in the MCFsyn reconciliation framework. In this manner, each participant customizes its MCF according to its local set cardinality.

Unlike homogeneous MCFs, the candidate buckets in heterogeneous MCFs with unequal lengths are not the same. This feature disables the proposed aggregation algorithm. Thus, we redesign the data structure of MCF so that the candidate buckets are derived out by the fingerprint. Given an element x with fingerprint η_x, the two candidate buckets are calculated as: $h_1(x) = hash(\eta_x)\%m$ and $h_2(x) = (h_1(x) \oplus \eta_x)\%m$. In this manner, when inserting fingerprints from two MCFs into the aggregated MCF, the candidate buckets can be simply determined by the fingerprints themselves. The impact of this alteration is theoretically detailed in [23]. For a participant P_i, the capacity of its MCF is determined as $m_i \cdot b_i \approx \alpha \cdot |S_i|$, where m_i, b_i are the number and capacity of buckets in MCF_i respectively, and $\alpha \geq 1$ is a constant coefficient.

Given MCF_i and MCF_j, the aggregation algorithm is also altered correspondingly. A new MCF MCF_o is initialized with capacity $m_o \times b_o = \alpha \times |S_i \cup S_j|$. The value of $|S_i \cup S_j|$ can be evaluated by counting the total number of distinct fingerprints in MCF_i and MCF_j. Thereafter, the fingerprints in both MCF_i and MCF_j, as well as the associated mark fields, are inserted into MCF_o one by one. The time-complexity is thereby increased from $O(mb)$ (for homogeneous MCFs) to $O(m_i b_i + m_j b_j)$.

Another problem of using heterogeneous MCFs is that transmitting MCFs with the MST may not be the optimal choice anymore. The reason is that transmitting the most lengthy MCF all along with the MST is not economic. To generate the optimal transmission route, one has to traverse all the n^{n-2} possible spanning trees [30] in the reconciliation group graph. This is certainly computation-intensive. Therefore, there is a trade-off between the communication cost and the computation complexity here. A proper spanning tree can surely lower the communication cost of MCFsyn, yet determining such a tree will occupy computation resources. For a bandwidth-scarce situation (such as wireless sensor networks), saving the bandwidth is the first priority; thus choosing the best spanning tree is worthwhile. By contrast, when the network resource is abundant, it may be not advisable to spend time to derive out the optimal spanning tree.

2) Dynamic Reconciliation Participants: In the design of the protocol we assumed the set of participants is fixed. In reality, participants may join in or depart unpredictably. In the reconciliation group graph, if a leaf participant of the construct MST departures, there is no impact to the rest of the MST. The corresponding bit in the mark field of MCF will be removed or ignored. On the other hand, when a non-leaf (internal) participant of the MST leaves, the MST is divided into multiple connected components. In that case, ***MCFsyn has to reconstruct a new MST***. This can be achieved based on the following cut property and corollaries.

Theorem 1: **(Cut property of MST [31])** Let $G(V,E)$ be a graph, $(X,V-X)$ be a cut of G, and edge e be the only minimum cost edge that crosses the cut $(X,V-X)$. Then every minimum spanning tree contains the edge e.

Corollary 1: Let $G(V,E)$ be a graph, T be an MST of G, $\hat{G}(\hat{V},\hat{E})$ (with $\hat{V}\subseteq V$ and $\hat{E}\subseteq E$) be a connected subgraph of G. $\hat{T}$ is a subtree of T which covers all the participants in $\hat{G}$. Then $\hat{T}$ is an MST of the graph $\hat{G}$.

Proof: Consider the cut $(\hat{G},G-\hat{G})$ and the MST T, then according to Theorem 1, we have $T=\hat{T}\cup\overline{T}\cup\hat{e}$, where $\overline{T}$ and $\hat{e}$ are the subtree of T which covers all participants in the graph $G-\hat{G}$ and the minimum edge across $\hat{G}$ and $G-\hat{G}$. If $\hat{T}$ is not an MST of graph $\hat{G}$, then it can be replaced by an MST of a smaller weight in $\hat{G}$ to allow an MST of G with a weight smaller than T, in contradiction. This proves the correctness of Corollary 1. ■

Corollary 2: For a fully connected reconciliation group graph $G(V,E)$ and its MST T, when a participant P_i leaves the group, T may be split as multiple subtrees. Connecting these subtrees incrementally with the minimum edges without creating cycles among them generates a new MST for the group without P_i.

Corollary 2 is a natural result of Theorem 1 and Corollary 1. Therefore we omit its proof here. Based on this corollary, MCFsyn reconstructs the disconnected MST by adding the edges with the minimum weight between these disconnected components. Certainly, when adding these edges, MCFsyn must ensure that there is no cycle introduced into the MST.

When a new participant P_{n+1} joins into the reconciliation group and all weights associated with the introduced edges are larger than the weights in the existing MST, MCFsyn just adds the edge which connects P_{n+1} with any existing participants and has the minimum weight into the MST. Otherwise, MCFsyn has to run some existing algorithms to update the MST with the time-complexity of $O(n^{4/3}logn)$ [32]. Besides, the MCF data structure introduces one additional bit into its mark field to explicitly demonstrate the members of the set S_{n+1}.

V. Performance Analysis

A. Failures of Element Representation

The ability of a CF to represent a set without failures in element insertions is probabilistic. In a CF with m_i buckets and b_i slots in each bucket, there exists a threshold H_i, when $H_i \geq \frac{|S_i|}{m_i}$, the $|S_i|$ elements can be represented by this CF successfully with probability $1-o(1)$; otherwise, this CF fails to represent all the $|S_i|$ elements with probability $1-o(1)$. This threshold is detailed in previous work [26] [33] [34]. Furthermore, an upper bound of the probability that all the $|S_i|$ elements are successfully represented by the CF with parameters m_i and b_i is also presented in [26]. We denote this upper bound as $\Phi(m_i,b_i,|S_i|)$. The same bound applies also to MCF since its additional field does not affect the element representation. Pair-wisely, following constraints among the CF parameters m_i, b_i, f, $|S_i|$ given in [23], a lower bound of the probability that all $|S_i|$ elements are successfully represented by the CF can be derived. We denote this lower bound which is also affected by the fingerprint length f as $\Theta(m_i,b_i,|S_i|,f)$. Then the probability that all participants represent their set members with MCFs successfully is bounded as:

$$\prod_{i=1}^{n}\Theta(m_i,b_i,|S_i|,f) \leq p_r \leq \prod_{i=1}^{n}\Phi(m_i,b_i,|S_i|). \tag{3}$$

With these theories, the capacity of our MCF can also be designed such that all the elements in each participant can be represented successfully with a probability close to 1. However, we cannot guarantee that every element is represented eventually. Therefore, we analyze the consequence of such failures in our MCFsyn framework. A reconciliation participant P_i, might have three kinds of elements: 1) common elements S_i^c which exist in every participants; 2) exclusive elements S_i^e which are only held by P_i; and 3) partial elements S_i^p which only appear in part of these participants (in at least one other set).

For a common element, if the local MCF fails to represent it, it will be regarded as a missing element during the extraction step. As a result, a false positive error of reconciliation (identifying a common element as a different one) will occur. In the last step of reconciliation, P_i has to request this element from the participant with the highest priority, introducing unnecessary element transfer.

The failure of representing an exclusive element at P_i, on the other hand, leads to a false negative error of reconciliation (identifying a different element as a common one). As a consequence, P_i will not push this exclusive element to others, resulting in inaccurate reconciliation. As for a partial element, missing its information also causes a false positive error of reconciliation. The participant P_i will pull this element from a non-optimal participant. Besides, for those participants which assign P_i with the highest priority, they cannot fetch this partial element with the minimum communication overhead.

B. Impact of Hash Collisions

In the framework of MCF, two independent sets of hash functions are employed — one for fingerprint generation and one for candidate buckets derivation. Hash collisions are not avoidable. When diverse elements are assigned to the same candidate buckets, this hash collision is resolvable because each bucket has b slots. Therefore, we focus on hash collisions in the generation of element fingerprints.

If two elements (e.g., x and y) share the same fingerprint yet have totally different candidate buckets, the aggregation and distribution step will not be affected. However, a problem will occur during the extraction process (detailed in Section IV) because participants cannot distinguish them when they try to fetch these two elements. To resolve this dilemma, participants only pull x and y from the participants which don't have x and y simultaneously. On the other hand, if x and y share common affiliates, they will be reconciled according to the standard extraction and transmission steps.

When elements x and y share the same fingerprint and some common candidate bucket(s), the aggregation step and the thereafter extraction step have to handle this exception. During aggregation, adding the affiliation information of x to y will surely lead to inaccurate reconciliation, and vice versa. MCFsyn mitigates this issue conservatively by broadcasting x and y to all the participants and setting the corresponding mark fields to 1s. Specifically, during aggregation, if P_i finds this collision, the elements x and y will be pushed to others from P_i or (and) the participants which hold x or y. After that, in the overall MCF, the bits of the mark fields in the corresponding slots are all set to 1 to explicitly declare that these elements are already common elements. The later steps can be executed as they are designed, without the worry of conflicted elements.

Theorem 2: Consider a n-party set reconciliation using MCFsyn with parameters m, b and f, given the boarder length of MCF $\hat{m}$ calculated with the threshold H, as long as we choose $m \geq (1+\varepsilon)\hat{m}$ for some $\varepsilon > 0$, the MCFsyn reconciles all the elements from $S = \cup_i S_i$ correctly with probability:

$$p = O(1-o(1)) \cdot 2^{-|S|f} \cdot \prod_{i=0}^{|S|-1} (2^f - i). \tag{4}$$

Besides, the reconciliation takes $2(n-1)$ steps with $2(n-1)$ total messages of size $O(|\cup_i S_i|)$ to identify the differences.

Proof: For a successful MCFsyn reconciliation, all the elements should be correctly represented and there must have no fingerprint collisions. According to the theory presented in [26] [33] [34], when $m \geq (1+\varepsilon)\hat{m}$ for some $\varepsilon > 0$, the MCF represents all elements in S successfully with probability $O(1-o(1))$. Given the number of bits in each fingerprint as f, the probability of collision-free fingerprint generation is $2^{-|S|f} \cdot \prod_{i=0}^{|S|-1} (2^f - i)$. Consequently, Equ. 4 follows. Moreover, MCFsyn aggregates and distributes the MCFs along with the MST to and from the logic central relay. Each aggregation and distribution operation carries an MCF. The size of that MCF is decided by the number of elements in the union set. Therefore, the whole reconciliation takes $2(n-1)$ steps with $2(n-1)$ total messages of size $O(|\cup_i S_i|)$ to identify all the different elements among participants. ■

C. The space overhead of MCFs

In this subsection, we quantify the space overhead of representing the elements for reconciliation with MCFsyn.

Suppose a reconciliation group with n participants, and the total number of elements in the union set $S = \cup_i S_i$ is $|S|$, then the bits per element (bpe) after the aggregation step can be formulated as:

$$BPE_{MCF} = \frac{(n+f) \times b \times m}{|S|}, \tag{5}$$

where f, b and m are the parameters of MCFs to represent the union set S. With the given b and $|S|$, the value of m can be derived out by the threshold introduced in previous work [26] [33] [34]. In effect, MCF introduces a *mark* field with n bits into each slot to label which participant(s) this element belongs to. Each slot needs $n+f$ bits, and the total number of bits of the whole filter is $(n+f) \times b \times m$. This confirms Equ. 5. Note that, if homogeneous MCFs are deployed, they share the same capacity yet unequal $|S_i|$. Therefore, the bpe of participant P_i at the representation step is $\frac{(n+f) \times b \times m}{|S_i|}$. By contrast, when heterogeneous MCFs are used as stated in Section IV-C, then the bpe of participant P_i at the representation step is $\frac{(n+f) \times b \times m_i}{|S_i|}$.

An extended problem is how to determine the value of $|S|$ before reconciliation. It can be abstracted as a *cardinality estimation* problem. There are a family of solutions [35], such as LogLog [36], SuperLogLog [37], HyperLogLog [37], HyperLogLog++ [38], MinCount [39], AKMV [40], etc. They usually rely on the bit-mapping, hashing, sampling techniques to derive out the cardinality of a given set. In our case, undoubtedly, to implement these methods among multiple reconciliation participants, multiple rounds of communications are necessary.

D. Communication cost of MCFsyn

In this paper, we measure the communication cost over a link as the product of the size of the carried message by the link weight (described in the reconciliation graph). The total communication cost is given by the summation over all links.

Theorem 3: Given the size of the sketch data structure, when the different elements can only be sent from their affiliates before reconciliation, MCFsyn achieves the *minimum communication cost* of multi-party set reconciliation.

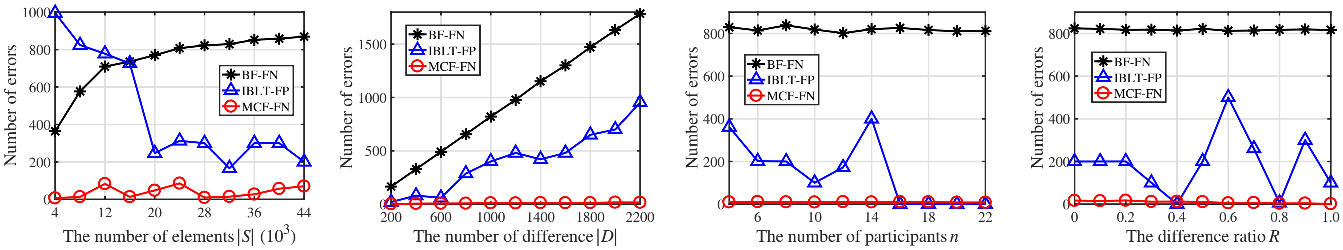
Proof: The communication cost of MCFsyn is caused by exchanging the MCFs (the aggregation and distribution steps) and transferring different elements (the transmission step) among the participants. Let C_a, C_d, C_e, and C_m denote the communication cost of sketch aggregation, sketch distribution, exclusive element transfer, and missing element transfer, respectively. The overall communication cost of MCFsyn (denoted as C_o) can be calculated as:

$$C_o = \; C_a + C_d + C_e + C_m. \tag{6}$$

Assuming the employed sketch data structure is B_s bits, then we have:

$$C_a = B_s \cdot \sum_{e_{i,j} \in E_a} w_{i,j} \quad \text{and} \quad C_d = B_s \cdot \sum_{e_{i,j} \in E_d} w_{i,j}, \tag{7}$$

where E_a is the set of edges which get involved with the aggregation step and E_d is the corresponding set for the distribution step. Note that both the aggregation and distribution steps need to cover all participants. Moreover, C_a/B_s and C_d/B_s



(a) The accuracy when $|S|$ varies. (b) The accuracy when $|D|$ varies. (c) The accuracy when n varies. (d) The accuracy when R varies.

Fig. 4. The multi-party reconciliation accuracy with diverse parameter settings.

are the sums of weights for all involved edges. Therefore the minimal C_a/B_s and C_d/B_s naturally fit the definition of a minimum spanning tree. Hence, the communication cost $C_a + C_d$ is minimized.

MCFsyn implements diverse transmission strategies for the exclusive elements and missing elements to save bandwidth. Specifically, when the exclusive elements are pushed to other participants along with the MST tree, we have:

$$C_e = \sum_{x \in D_E} B_x \cdot \sum_{e_{i,j} \in MST} w_{i,j}, \tag{8}$$

where x is an exclusive element and B_x is the number of bit of the element x. Consequently, when distributing the exclusive elements with the MST, MCFsyn minimizes the cost C_e. As for C_m, note that in MCFsyn, for missing elements with multiple holders, the local participant just fetches the content from its nearest participants. This mechanism naturally minimizes the cost C_m. Thus, the communication cost of transferring different elements is also optimal in MCFsyn. ■

VI. Evaluation

In this section, we compare MCFsyn with other methods for multi-party set reconciliation, including the BF-based method and the IBLT-based method. All the experiments are conducted on a host with 8 GB RAM and 3.4 GHz CPU. Note that the element content has merely impact on the reconciliation performance, thereby we generate random strings with diverse lengths as set elements. The metrics include the number of false positives, the number of false negatives, and the communication cost of reconciliation.

A. Reconciliation Accuracy

We quantify the false positives (FPs) and false negatives (FNs) of reconciliation with different parameter settings. The considered parameters include the number of elements in the union set S, denoted as $|S|$, the number of different elements $|D|$ (appear in some but not all sets), the ratio between $|D_E|$ (appear in only one of the sets) and $|D|$ denoted as R and the number of participants n. The default parameter setting is $|S|$=28,000 with $|D|$=1,000, R=0.5 and, n=10. Given the same ***bits per element*** for each data structure, we alter the above parameters respectively and plot the results in Fig. 4.

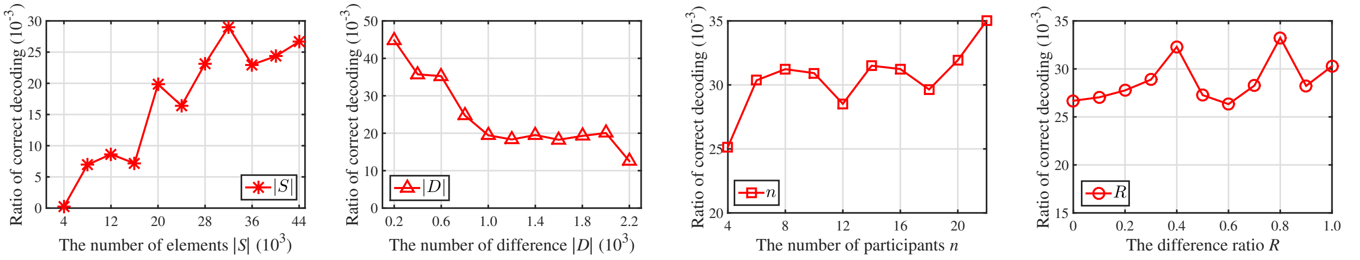
We also observe that the BF-based method and our MCF-based method incur no false positive errors; the IBLT-based method, by contrast, leads to no false negative errors. In effect, BF relies on the membership query operations to determine the different elements among participants. MCFsyn traverses the overall MCF to identify the different elements. Therefore they never suffer from false negative errors, once the elements are represented successfully. The IBLT, on the contrary, tries to list the different elements from the subtracting result which only contains the information of different elements. Consequently, IBLT may fail to list these elements but will never identify a common element as a different one.

We first set the total number of elements $|S|$ between 4,000 to 44,000. As depicted in Fig. 4(a), *BF-FN* shows a significant growth from 364 to 869. The reason is that larger $|S|$ means more membership queries, thus increasing the risk of false negatives of set reconciliation. On the contrary, the IBLT-based method leads to less false positives when $|S|$ grows. Intrinsically, more elements require more cells to represent them. In the generated IBLT after subtracting, more cells increase the probability of successful decoding. Obviously, our MCFsyn has the least false negatives. These false negatives are caused by the failures of representing elements. Certainly, by lengthening the MCFs, we can generate an even better reconciliation accuracy.

We then change the number of different elements $|D|$ from 200 to 2,200. As plotted in Fig. 4(b), when $|D|$ grows, both BF and IBLT experience increasing reconciliation errors. When $|D|$ increases, more elements in D may be identified as common ones by BFs. IBLT need to decode more elements from the subtracting result; thereby its risk of unsuccessful decoding rises up significantly. As for MCFsyn, we only observe a few false negatives in our experiments, rising from 1.8 to 16. The reason for such an increasing trend is that more element representation failures may happen upon the different elements when $|D|$ grows.

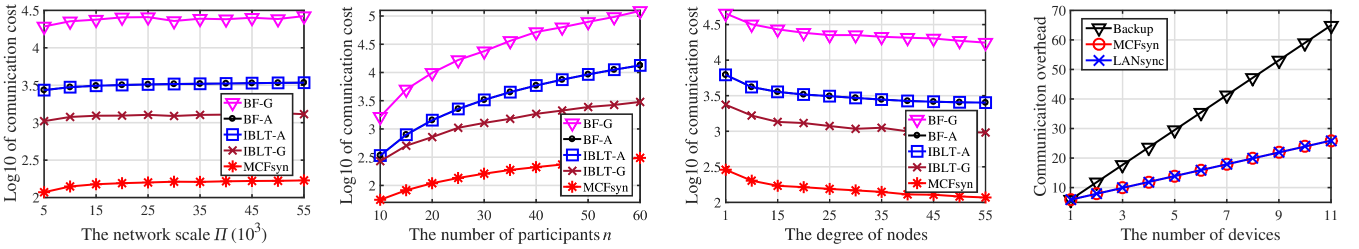
Lastly, we vary the ratio between $|D_E|$ and $|D|$ from 0 to 1. As specified in Fig. 4(d), BF still causes the most false negatives, fluctuating around 818. The number of false negatives resulted by MCF decreases from 17 to 1.33. In fact, a larger value of $|D_E|$ lowers the number of elements in each participant. Consequently, the participants can represent their local sets with MCF successfully with a higher probability. This explains the decreasing curve of *MCF-FN* in Fig. 4(d). IBLT, on the contrary, shows its instability and still incurs much more errors than our MCF.

Additionally, we also note that the IBLT-based method [16]



(a) Correct decoding when $|S|$ varies. (b) Correct decoding when $|D|$ varies. (c) Correct decoding when n varies. (d) Correct decoding when R varies.

Fig. 5. The fraction of correctly decoded different elements by the IBLT based reconciliation method.



(a) The cost when Π varies. (b) The cost when n varies. (c) Cost when node degree varies. (d) Cost with diverse strategies.

Fig. 6. The communication cost of multi-party set reconciliation for diverse methods.

may fail to tell the affiliation of these elements precisely. Therefore, Fig. 5 presents the exact fractions of the decoded different elements with correct affiliations by IBLTs when the parameters change. It is clear that IBLT can seldom judge the affiliation of the different elements correctly (the fraction of such correct decoding is at the level of 10^{-2}). As suggested by Mitzenmacher and Pagh [16], IBLT leverages the parity of the corresponding bit in the ID field of each cell to represent the affiliation of each stored element. However, when the i^{th} set has multiple elements mapped into this cell, the parity of the i^{th} bit in the ID field may fail to tell the affiliation of these elements. The absence of the exact affiliation information of the recorded elements surely results in inaccurate set reconciliation. This is also why we introduce the completeness feature into MCF .

Given the default setting, i.e., $|S|$=28,000, $|D|$=1,000, R=0.5 and n=10, we vary the four parameters respectively and record the generated ratio of correct decoding in IBLT. As depicted in Fig. 5(a), when $|S|$ increases from 4,000 to 44,000, more elements in D are decoded correctly. The behind reason is that, the increasing number of elements generates a longer IBLT vector, which creates a higher probability of successful decoding. However, in Fig. 5(b), the growth of $|D|$ decreases the fraction of successful IBLT decoding. The decrement is expected since it can be more difficult to search a "pure" cell for decoding when IBLT stores more different elements.

Additionally, as reported by Fig. 5(c), when the number of reconciliation participants n grows from 4 to 24, the fraction of correct decoding experiences a slight growth. With the given total number of elements and growing number of participants, each participant holds shrinking number of elements. Consequently, the parity of the corresponding bit in the ID field can correctly represent the affiliation of elements with a higher probability. The difference ratio R, on the contrary, has limited impact on the IBLT decoding accuracy. Therefore, the fraction of correct decoding in Fig. 5(d) fluctuates around 29×10^{-3}. Note that, the curves in Fig. 5 are unstable to some extent. The instability is caused by the decodings which end far early before all the different elements are listed. IBLT's unstable performance further limits its usage in multi-party reconciliation scenarios.

B. Communication Cost

During reconciliation, the participants need to exchange their local sketches with others. Therefore, the communication overhead of transferring these sketches is a joint result of the employed data structure and the transfer scheme. MCFsyn relies on the MST to aggregate and distribute the MCFs. By contrast, other reconciliation methods launch the all-to-all transmission or the gossip traffic (denoted as the suffix *-A* and *-G* respectively in the legends of Fig. 6) within the reconciliation group to exchange their sketches.

We evaluate the communication cost of transferring the sketch data structures in a random regular graph (a widely used topology in computer networks), with the consideration of three main parameters, i.e., the network scale Π, the number of reconciliation participants n, and the degree of nodes in the network. We let Π=30,000, n=30 and degree=20 by default and then vary them respectively to compare the communication cost of distinct combinations of data structures and transmission schemes. In our evaluations, we assume that each of the BF, IBLT, and MCF occupies the same space (normalized as 1 unit for simplicity) to represent each set.

As plotted in Fig. 6(a), when the underlying network scale grows from 5,000 to 55,000, all methods lead to increasing communication cost of transferring their sketches. Note that transmitting BF and IBLT with the all-to-all strategy incur the same communication cost. In this strategy, all participants send their local sketch to others. Therefore the total communication cost is $\sum_{i=1}^{n}\sum_{j=1}^{n} L_{i,j}$, where $L_{i,j}$ is the shortest path length from participant i to participant j. To gossip these sketches, $\log n$ rounds of transmissions are required on average. Each participant merges the received IBLTs as a single IBLT. The generated IBLTs are exchanged in the next round of gossip transmission. However, BFs cannot be merged. The participants have to send all the received BFs out in the next round of gossip transmission. This explains why the curve *BF-G* is much higher than *IBLT-G* in Fig. 6. In contrast, MCFsyn causes the least communication cost by aggregating and distributing the MCFs with MST. Beside of the MST, just like IBLT, the mergeability of MCF also highly contributes to such an excellent performance.

When n varies from 10 to 60 (in Fig. 6(b)) we observed the increase of communication cost for all measured methods. With more reconciliation participants, we have to transfer more sketches with the underlying network. Still, gossiping BFs results in the most communication cost, while our MCFsyn incurs one or two orders less communication cost than others. On the contrary, as presented in Fig. 6(c), when the node degree in the network gets larger, the communication cost for all the methods decreases constantly. The increased degree shortens the distance between all participants, thereby lowers the cost. MCFsyn still outperforms others in a large scale.

We further consider the communication cost of transmitting different elements. Note that, due to the unacceptable reconciliation accuracy or (and) the incapability of telling the affiliation of different elements, both the BF and IBLT enabled methods can be impractical. Consequently, we only compare the caused communication cost of our MCFsyn protocol, the backup strategy and the LAN sync strategy [41] in cloud services. The network topology is illustrated in Fig. 7. The backup strategy is widely used in current personal file cloud storage applications, such as Dropbox, Google Drive, OneDrive, etc. In this backup strategy, each local device uploads its data to the remote cloud for backup (usually three replicas in the cloud). Then the cloud pushes its data to the devices for consistency. We also note that, Dropbox has a “LAN sync” option for users. When a file is edited by a device, the updated version is first uploaded to the cloud server for sure. Thereafter, instead of pulling updates from the cloud server, the devices will try to obtain the updates from the devices in the same LAN first. By doing so, unnecessary cloud-device communications are avoided significantly.

As depicted in Fig. 6(d), this kind of communication overhead of backup strategy and our MCFsyn protocol increases linearly with the growth of the number of local devices. However, the backup strategy costs much more than our MCFsyn protocol. Specifically, in our evaluation, we suppose that the local devices are interconnected with an access point. According to [42] the AS level path length of the Internet is 3.9. For simplicity, we suppose the hops from the access point to a cloud rack is approximated as 3.9. Therefore, fetching (uploading) a data block from (to) the backup server costs 5.9 hops. Given the same volume of different elements to reconcile (normalized as 1 unit), the backup strategy causes 5.9·#device communication cost. The reason is that each different element has to be sent to the cloud first. After that, the cloud will push that element to the rest of the devices. By contrast, this kind of communication cost in MCFsyn is 2·(#device−1)+5.9. Note that the addend 5.9 indicates the cost of transferring different elements to the cloud; while 2·(#device−1) is the overhead of transmitting the different elements from its host to other devices by using the access point as a relay. As for LAN sync, in this network setting, it realizes the same communication overhead of transmitting different elements as MCFsyn does, since all the devices lie in a single LAN and the different elements can be fetched locally.

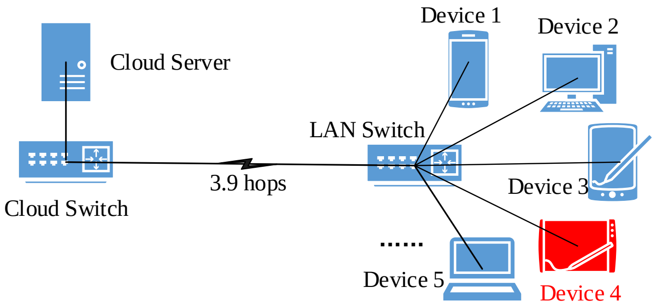


Fig. 7. A typical network topology of personal cloud storage system, wherein the personal devices are interconnected with a LAN switch, i.e., an access point. A device (device 4 in this example) has 1 unit of new files to sync with others.

However, we argue that MCFsyn generally outperforms LAN sync from the following two aspects. First, in MCFsyn, participants always try to get the missing elements with minimal overhead; while this is not guaranteed in LAN sync since it fails to optimize inter-LAN reconciliations. For instance, if the devices in Fig. 7 belong to diverse LANs, the devices in a LAN can only fetch updated data from the remote cloud server but not other LANs even they may be geographically close. Second, the LAN sync has not changed the nature of the backup strategy in personal cloud storage. In other words, LAN sync is still centralized, while MCFsyn is fully decentralized. The cloud server still plays a central role in the whole system. It confirms the changes and initiates the sync process. What LAN sync does is only try to speed up the data distribution step. By contrast, in MCFsyn, all the participants are totally equal and self-organized. They decide and then push/pull the different elements all by themselves. Such a design philosophy brings both flexibility and fault-tolerance. The participants are allowed to join or leave dynamically. Even when the cloud-LAN links fail, the devices in Fig. 7 can still sync with each other with our MCFsyn protocol; but such a failure may be problematic in LAN sync.

C. MCFsyn in Blockchains

In a blockchain, the transactions are validated and propagated as blocks. A fundamental question is that the transactions should be synchronized before block propagation. Specifically, each peer in the blockchain maintains a *mempool*

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/TPDS.2021.3074440, IEEE Transactions on Parallel and Distributed Systems

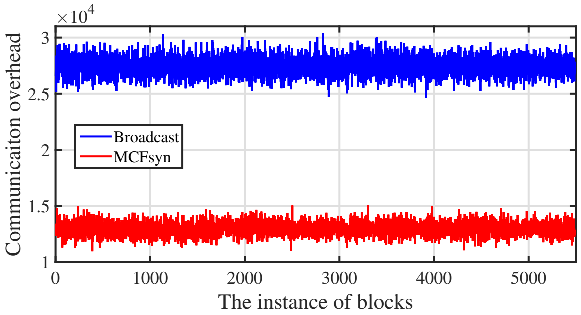
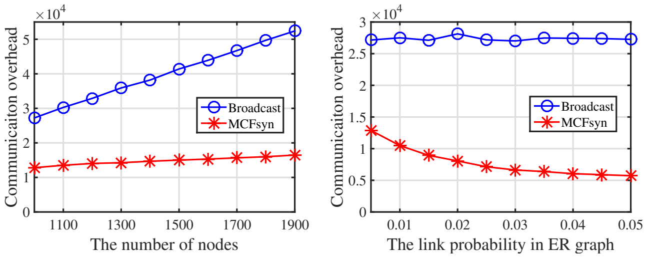


Fig. 8. The communication overhead of reconciling transactions in blockchain networks when $n = 1,000$ and $p = 0.05$.



(a) The cost when n varies.

(b) The cost when p varies.

Fig. 9. The communication cost of reconciling transactions in blockchain networks when n and p vary.

which stores unpackaged transactions. The problem can be abstracted as set reconciliation between *mempools*. The state-of-the-art method for such a scenario is called Graphene [4] which employs both BF [6] and IBLT [8] to identify the missing transactions in a peer. However, Graphene and its same kind only try to solve the two-party reconciliation problem, thus fail to achieve an overall optimization of communication overhead. Basically, Graphene needs to broadcast the newly formed block from the source to others, with the two-party synchronization strategy.

In this paper, we conduct simulations to compare MCFsyn with Graphene in blockchains with real Ethereum transactions [43]. We consider the first million on-chain blocks and extract more than 5,500 blocks that possess no less than 30 transactions. Thereafter, we try to propagate such blocks by reconciling their transactions. In our experiment, we generate ER graphs to simulate the underlying blockchain P2P networks and assign weights to the links to demonstrate the communication cost. We simplify the size of each transaction as 1 unit and then sum up the weights of utilized links to quantify the overall communication cost for network-wide consistency. For each node, we decide a random number in [0, 10] as its missing transactions. Our target is to reconcile such randomly chosen transactions for each block in the network.

As shown in Fig. 8, MCFsyn outperforms the broadcast strategy in current blockchain systems significantly. To be specific, when n=1,000 and the probability in ER graph is p=0.05, the overhead of MCFsyn ranges from 10,947 to 15,050, with an average value of 12,939. By contrast, the minimum, maximum, and average costs of the broadcast strategy are 24,600, 30,399, and 27,456, respectively. In other words, MCFsyn saves more than half of the communication overhead when transmitting the different transactions. The intrinsic reason is that a node in MCFsyn always tries to fetch its missing transactions with the minimum cost. However, for broadcast transmission, nodes just push/pull missing transactions from their neighbors without considering the cost.

Moreover, we quantify the communication cost of transmitting transactions in the blockchain network when the number of nodes n and the probability of links in ER graph p increase constantly. The results are presented in Fig. 9. Obviously, MCFsyn still outperforms the broadcast strategy significantly. When n grows, both the MCFsyn and the broadcast methods lead to increasing cost, since more links are employed to transmit the missing transactions. When p increases from 0.005 to 0.05, the communication cost decreases in MCFsyn while remains at a high level in broadcast strategy. With more links in the network, MCFsyn can search out a smaller MST with high probability; for broadcast, the introduced links may help to speed up the transmission but cannot decrease the communication cost.

As a summary, MCFsyn significantly outperforms other methods. Quantitatively, in our evaluations, MCFsyn generates 20x and 70x times fewer errors on average than the methods enabled by IBLT and BF with the same bits per element, respectively. Moreover, MCFsyn causes 21x and 7.9x times less communication cost on average than transferring BFs with the all-to-all scheme and exchanging IBLTs with the gossip protocol, respectively. Especially, the IBLT-based method can barely tell the affiliations of its decoded elements. MCFsyn is competent to discover the different elements with their correct affiliations so that these different elements can be synchronized to other participants with the optimal senders. Therefore, MCFsyn shows much less communication cost to deliver these different elements than the general backup strategy used in current cloud storage services. Besides, the trace-driven simulations indicate that MCFsyn can half the communication overhead of transmitting transactions, compared with the current broadcast-based reconciliation strategy.

VII. Discussion

In this section, we further discuss several issues about the MCFsyn protocol.

Evil nodes in the reconciliation group. In a distributed scenario, it is possible that evil nodes intend to steal information from the group or spread unwanted data within the group. Such an issue is beyond the scope of this paper. However, we think it is costly or uneasy to do so. First, the hackers have to pass the verification and authorization from the upper-level applications, before becoming a member of the reconciliation group. For enterprises or institutions which build their own private networks, they have no such worry since they are independent from the public Internet physically. Second, the network security modules in the network monitor the network consistently and provide security to the reconciliation group, to some extent. If the evil nodes attack or disturb the network, the defense facilities will act.

When to run the MCFsyn protocol. MCFsyn is a low-level protocol to synchronize content among multiple participants.

An upper-level question is when MCFsyn should be executed. In effect, there is a body of work that focuses on this field [44] [45]. For example, UDS proposes to batch the micro changes before synchronization to save bandwidth [44]. These strategies can potentially be applied to MCFsyn. In our case, an upper-level control scheme can be either update-driven or time-divided. For an update-driven policy, the MCFsyn is triggered once new elements are included or existing elements are edited. Alternatively, it is also advisable to prob the participants periodically and sync the updates in a batched fashion. Generally, the latter can is more bandwidth-friendly than the former, since reconciling the micro updates may cause the overuse of synchronization messages.

MCFsyn in extreme scenarios. MCFsyn is a general protocol for multi-party reconciliation and can be applied to reconcile elements no matter how the different elements distribute among the participants. When the participants have a skewed number of elements, following the MCFsyn steps can identify and transmit the different elements successfully. A possible optimization for this case is to choose the node with the most elements as its central relay, such that the others may use smaller MCFs for bandwidth saving purposes. Besides, in the scenario where the sets are totally or barely different, MCFsyn still works. Specifically, if the sets are almost the same, MCFsyn should still be employed to discover the different elements. By contrast, when they share no common elements, MCFsyn can still guide the participants to transmit the elements along with the MST tree for fast and economic reconciliation.

VIII. Conclusion and Future Work

In this paper, we present the MCFsyn protocol for multi-party set reconciliation in distributed scenarios. MCFsyn employs the MCF data structure as a sketch of each set, thereafter aggregates and distributes the MCFs with the underlying MST to minimize the communication cost. The aggregating-while-transmitting strategy further reduces the communication cost of exchanging MCFs. By differentiating the exclusive elements from missing elements, MCFsyn realizes an optimal strategy to transmit the different elements among the participants. The comprehensive evaluations indicate that MCFsyn significantly outperforms its same kind in terms of both reconciliation accuracy and communication cost.

Our future work is mainly two-fold. First, we will implement MCFsyn in real systems, such as open-source blockchain platforms Ethereum [46], Hyperledger [47], Corda [48], and private cloud storage systems such as OwnCloud [49], Nextcloud [50], Seafile [51], etc. As far as we know, these blockchain and cloud storage systems mainly rely on a two-party reconciliation scheme [4] to synchronize transactions before block construction. Our MCFsyn may help to speed up this process with less communication overhead. Second, we would like to consider the multi-party reconciliation of multi-sets wherein elements are allowed to have multiple instances. In such cases, both the element content and multiplicity can cause differences; thus identifying and transmitting such elements can be more challenging.

Acknowledgment

The authors thank all the anonymous reviewers for their insightful feedback. Besides, this work was supported in part by the National Key Research and Development Program of China under grant 2018YFB1800203, in part by the National Natural Science Foundation of China under Grant 62002378, and in part by the Research Funding of NUDT under Grant ZK20-3.

References

[1] 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.

[2] V. Stefano, L. Vanbever and O. Bonaventure, “Opportunities and research challenges of hybrid software defined networks,” *in Proc. of ACM SIGCOMM,* 2014.

[3] B. Maggs and R. Sitaraman, “Algorithmic nuggets in content delivery,” *ACM SIGCOMM Computer Communication Review,* vol. 45, no. 3, pp. 52-66, 2015.

[4] A. Ozisik, B. Levine, G. Bissias, G. Andresen, D. Tapp and S. Katkuri, “Graphene: Efficient interactive set reconciliation applied to blockchain propagation,” *in ACM SIGCOMM,* 2019.

[5] 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.

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

[7] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, “What’s the difference?: Efficient set reconciliation without prior context,” *in Proc. of ACM SIGCOMM*, 2011.

[8] M. T. Goodrich and M. Mitzenmacher, “Invertible Bloom lookup tables,” *in Proc. of Allerton Conference,* 2011.

[9] D. Chen, C. Konrad, K. Yi, W. Yu, and Q. Zhang, “Robust set reconciliation,” *in Proc. of ACM SIGMOD,* 2014.

[10] L. Luo, D. Guo, J. Wu, O. Rottensreich, Q. He, Y. Qin, and X. Luo, “Efficient multiset synchronization,” *IEEE Transactions on Networking,* vol. 25, no. 2, pp. 1190-1205, 2017.

[11] G. Karpovsky, and B. Levitin, “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] B. Fan, D. Andersen, M. Kaminsky, and M. Mitzenmacher, “Cuckoo filter: Practically better than Bloom,” *in Proc. of ACM CoNEXT,* 2014

[14] Z Zhu, A Afanasyev, “Let’s chronosync: Decentralized dataset state synchronization in named data networking,” *in Proc. of IEEE ICNP,* 2013.

[15] T. Chekam, E. Zhai, Z. Li, Y. Cui, and K. Ren, “On the synchronization bottleneck of OpenStack Swift-like cloud storage systems,” *in Proc. of IEEE INFOCOM,* 2016.

[16] M. Mitzenmacher, R. Pagh, “Simple multi-party set reconciliation,” *Distributed Computing,* vol. 31, no. 6, pp. 441-453, 2018.

[17] A. Boral, M. Mitzenmacher, “Multi-party set reconciliation using characteristic polynomials,” *in Proc. of Allerton Conference,* 2014.

[18] F. Hao, M. Kodialam, T. V. Lakshman, and H. Song, “Fast dynamic multiset membership testing using combinatorial Bloom filters,” *in Proc. of IEEE INFOCOM,* 2009.

[19] M. Mitzenmacher, P. Reviriego, and S. Pontarelli, “OMASS: One memory access set separation,” *IEEE Transactions on Knowledge and Data Engineering,* vol. 28, no. 7, pp. 1940-1943, 2016.

[20] H. Dai, Y. Zhong, A. Liu, W. Wang, and M. Li, “Noisy Bloom filters for multi-set membership testing,” *in Proc. of ACM SIGMETRICS,* 2016.

[21] D. Yang, D. Tian, J. Gong, S. Gao, T. Yang, and X. Li, “Difference Bloom filter: A probabilistic structure for multi-set membership query,” *in Proc. of IEEE ICC,* 2017.

[22] T. Yang, D. Yang, J. Jiang, S. Gao, B. Cui, L. Shi, and X. Li, “Coloring embedder: A memory efficient data structure for answering multi-set query,” *in Proc. of IEEE ICDE,* 2019.

[23] D. Eppstein, “Cuckoo filter: Simplification and analysis,” *arXiv preprint,* arXiv:1604.06067, 2016.

[24] M. Mitzenmacher, S. Pontarelli, and P. Reviriego, “Adaptive Cuckoo filters,” *in Proc. of SIAM ALENEX,* 2018.

[25] H. Chen, L. Liao, H. Jin, and J. Wu, "The dynamic cuckoo filter," *in Proc. of IEEE ICNP,* 2017.

[26] L. Luo, D. Guo, O. Rottenstreich, R. T. B. Ma, X. Luo, and B. Ren, "The Consistent cuckoo filter," *in Proc. of IEEE INFOCOM,* 2019.

[27] N. Nguyen, P. Tsigas, "Lock-free cuckoo hashing," *in Proc. of IEEE ICDCS,* 2014.

[28] A. Breslow and N. Jayasena, "Morton filter: fast, compressed sparse cuckoo filters," *The Very Large Data Bases Journal,* vol. 29, pp. 731-754, 2020.

[29] M. Wang, M. Zhou, S. Shi, and C. Qian, "Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters," *In Proc. of VLDB,* 2019.

[30] Y. Wu and K. Chao, "Spanning trees and optimization problems", *Chapman and Hall/CRC,* 2004.

[31] J. Kleinberg and E. Tardos, "Algorithm design," *Pearson Addison Wesley,* 2006.

[32] M. Henzinger, V. King, "Maintaining minimum spanning trees in dynamic graphs," *in Proc. of ICAPL,* 1997.

[33] M. Dietzfelbinger, A. Goerdt, M. Mitzenmacher, A. Montanari, R. Pagh, and M. Rink, "Tight thresholds for cuckoo hashing via XORSAT," *in Proc. of ICALP*, 2010.

[34] N. Fountoulakis, M. Khosla, and K. Panagiotou, "The multiple orientability thresholds for random hypergraphs," *in Proc. of ACM-SIAM SODA,* 2011.

[35] H. Harmouch, F. Naumann, "Cardinality estimation: an experimental survey," *in Proc. of VLDB*, 2017.

[36] M. Durand, P. Flajolet, "LogLog counting of large cardinalities," *in Proc. of European Symposium on Algorithms,* 2003.

[37] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm," *in Proc. of Discrete Mathematics and Theoretical Computer Science,* 2008.

[38] P. J. Haas and L. Stokes, "Estimating the number of classes in a finite population," *Journal of the American Statistical Association,* vol. 93, no. 444, pp. 1475–1487, 1998.

[39] F. Giroire," Order statistics and estimating cardinalities of massive data sets," *Discrete Applied Mathematics,* vol. 157, no. 2, pp. 406–427, 2009.

[40] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," *in Proc. of SIGMOD,* 2007.

[41] What is LAN sync? https://help.dropbox.com/installs-integrations/sync-uploads/lan-sync-overview.

[42] B. Edwards, S. Hofmeyr, G. Stelle, and S. Forrest, "Internet topology over time," *arXiv preprint,* arXiv:1202.3993, 2012.

[43] Ethereum On-chain Data, http://xblock.pro/xblock-eth.html.

[44] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Zhao, C. Jin, Z. Zhang, Y. Dai, "Efficient batched synchronization in dropbox-like cloud storage services," *in Proc. of ACM/IFIP/USENIX Middleware,* 2013.

[45] L. Caviglione, M. Podolski, W. Mazurczyk, "Covert channels in personal cloud storage services: The case of Dropbox," *IEEE Transactions on Industrial Informatics,* vol.13, no.4, pp. 1921-1931, 2017.

[46] Ethereum, https://ethereum.org/en/.

[47] Hyperledger, https://www.hyperledger.org/.

[48] Corda, https://www.corda.net/.

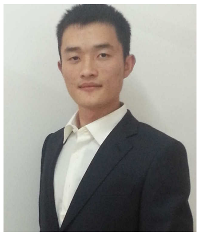
[49] OwnCloud, https://owncloud.com/.

[50] Nextcloud, https://nextcloud.com/.

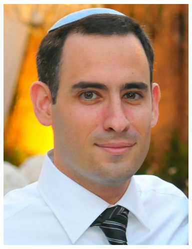
[51] Seafile, https://www.seafile.com/en/home/.

Lailong Luo received his B.S. and M.S. Ph.D degree at the College of Systems Engineering from National University of Defence Technology, Changsha, China, in 2013, 2015, and 2019 respectively. He is currently a lecturer in School of Systems, National University of Defense Technology, Changsha, China. His research interests include data structure and distributed networking systems.

Deke Guo Deke Guo received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of Systems Engineering, National University of Defense Technology His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.



Yawei Zhao Yawei Zhao is now a Ph.D candidate in School of Computer, National University of Defense Technology. He received his B.E. degree and M.S. degree in Computer Science from the National University of Defense Technology, China, in 2013 and 2015, respectively. His research interests include numerical optimization algorithms, pattern recognition and machine learning. He is also interested in data structures and algorithms.



Ori Rottenstreich is an assistant professor at the Department of Computer Science and the Department of Electrical Engineering of the Technion, Haifa, Israel. He is also the chief scientist of Orbs. His main research interest is computer networks and blockchain technologies. In 2015-2017 he was a Postdoctoral Research Fellow at the Department of Computer Science, Princeton University. Earlier, he received the BSc in Computer Engineering (summa cum laude) and PhD degree from the Technion in 2008 and 2014, respectively.

Richard T.B Ma received the Ph.D. degree in Electrical Engineering in May 2010 from Columbia University, New York. During his Ph.D. study, he worked as a research intern at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, and Telefonica Research, Barcelona, Spain. He is currently a Research Scientist in Advanced Digital Science Center, University of Illinois, USA, and an Assistant Professor in School of Computing at National University of Singapore. His research interests include distributed systems and network economics.

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.