BCE: a Privacy-preserving Common-friend Estimation Method for Distributed Online Social Networks Without Cryptography

Yongquan Fu and Yijie Wang
National Key Laboratory for Parallel and Distributed Processing, College of Computer Science
National University of Defense Technology
Hunan province, China, 410073
Email: yongquanf@nudt.edu.cn, wangyijie@nudt.edu.cn

***Abstract*—Distributed online social networks (DOSN) have emerged recently. Nevertheless, recommending friends in the distributed social networks has not been exploited fully. We propose BCE (*B*loom Filter based *C*ommon-Friend *E*stimation), a scalable and privacy-preserving common-friend estimation scheme that estimates the set of common friends without the need of cryptography techniques. First, BCE denotes each user using the identifiers created by the Peer-to-Peer underlay that are robust against the dictionary attacks. Second, BCE uses a Bloom filter to represent a friend list for scalability. Third, BCE estimates common friends of two users using the intersection of Bloom filters computed by one of their common friends, which ensures the privacy of friend lists against unknown users. Our privacy analysis shows that BCE hides the privacy of each user with a high probability. Simulations over real-world social-network data sets confirms that BCE is both accurate and scalable.**

I. Introduction

Distributed online social networks e.g., Safebook [1], LotusNet [2], Cuckoo [3], diaspora [4], Peerson [5], Vis-a-Vis [6] have been proposed to better protect users' privacy based on decentralized infrastructures. Friendship relationships in the distributed social networks are based on real-world friends. People independently store their personal data into decentralized machines, and enforce strict access policies to their data for friends.

Most decentralized social networks enforce that the communication between users must follow the links of friends in order to establish the trust on data communications. As a result, when people want to publish or query data on the decentralized social networks, the messages are forwarded between friends in a hop-by-hop manner, in order to hide the source and destination users. To do that, a Peer-to-Peer substrate is usually constructed to forward the messages.

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2011CB302601; the National Natural Science Foundation of China under Grant No.60873215; the Natural Science Foundation for Distinguished Young Scholars of Hunan Province under Grant No. S2010J5050; Specialized Research Fund for the Doctoral Program of Higher Education under Grant No.200899980003; the National High Technology Research and Development Program of China (863 Program) under Grant No.2011AA01A202.

An open question for the distributed social networks is how to recommend future friends for users. To do that, we need to quantify the degree of friendship between two users that have not been friends. A common practice by centralized social networks is to use the **common conditions** to measure the possibility of being friends, such as the list of common friends, common social groups or common interests. The common conditions between people are natural reasons for establishing friendships confirmed by pioneering researchers [7], which are also simple to be implemented assuming the complete social graph is stored in a centralized site.

In this paper, we choose the common friends as an example of the common conditions. Measuring pairwise common friends in a decentralized setting, is however, a challenging task, since the friends of a user is sensitive personal information, which should be protected against curious users. Furthermore, measuring common friends should scale well with increasing friend lists.

In the decentralized setting, we are aware that Talash [8] directly computes the common items of two friend lists in order to obtain the common friends. Unfortunately, transmitting the friend lists does not scale well and exchanging friend lists to unknown users may leak the privacy of friends' information, which is not desirable for the DOSNs. Therefore, we need a scalable and dynamic common-friend estimation scheme that protects the privacy of friend lists and scales well with increasing friends.

We propose a distributed common-friend estimation scheme BCE (*B*loom Filter based *C*ommon-Friend *E*stimation) that estimates common friends scalably without disclosing the content or the size of the friend list of a user.

Our key insight is that, since most Peer-to-Peer underlays of the DOSN allocate each user a unique **identifier** created by cryptographic hash functions like SHA-1 hash functions from a wide range of space (e.g., 160-bit strings), these identifiers are robust against dictionary attacks by curious users. For example, for 160-bit strings created by SHA-1 hash functions, a curious user must iterate 2^{160} strings to completely pick out the friends of a user, which is computationally difficult for off-the-shelf machines. As a result, The identifiers hide the privacy of users, by decoupling the DOSN from the real-world identity of a person.

BCE uses the identifiers to represent each friend to mitigate the dictionary attacks. Unfortunately, the size of the friend list may still leak the personal information that may reflect the social popularity of a user, which may be sensitive for users. To hide the size of the friend list and scale well with increasing friends, we use the well-known Bloom filter to represent the friends. Due to the dynamics of friends, each user independently maintains his (or her) Bloom filter to control the **false positives** of the Bloom filters that may report a user that is not in the set to be in the set.

Unfortunately, sending the Bloom filters to unknown users may still leak personal information. For example, a user can query a Bloom filter using a set of crawled identifiers from the DOSN. Therefore, some items in the friend list may be found by the query process. However, completely obtaining the items represented by a Bloom filter is infeasible, due to the resilience of dictionary attacks by the identifiers.

In order to hide the privacy leakage of the Bloom filter, we could use the cryptographic techniques to encrypt the Bloom filters. However, the computation and the communication overhead will increase significantly. In this paper, we use the intersection of Bloom filters to represent the common items and hide the non-common items. Our theoretical analysis shows that the intersection of Bloom filters hides the presence of any non-common items. A user that is not the common friend of two users is always able to claim he (or she) is not in the friend list of a user.

To compute the intersection of Bloom filters for users A and B without disclosing the Bloom filters to users that are not friends of A or B, we *select common friends of two users A and B to be responsible for the computation*. Since each user allows his (or her) friend to visit his (or her) friend list, a user can transmit his (or her) Bloom filter to his (or her) friend without leaking the friend privacy.

Suppose that two users A and B are not friends and have common friends. We can see that users A and B must be **friends of friends**, which implies that these two users are two hops away on the DOSN. Fig 1 shows an example. To estimate common friends, each user periodically pushes his (or her) Bloom filter to his (or her) friends. Furthermore, each user periodically computes the intersection of two Bloom filters for any pair of friends and pushes the intersection to this friend pair. Finally, when a user A receives the intersection of his (or her) Bloom filter with another Bloom filter of user B, user A can query the intersection to obtain the estimated common friends. Simulation results show that BCE can accurately estimate common friends with modest bandwidth cost, which indicate that BCE is feasible for common-friend measurements on a highly decentralized environment.

The rest of the paper is organized as follows. Section II summaries related work on common-friend measurements. Section III introduces the background information for the common-friend measurement. Section IV presents the basic idea of BCE. Section V shows the Bloom filter based friend representation. Section VI shows how to compute the intersection of Bloom filters. Section VII analyzes the privacy protection of the Bloom filters and the intersection of Bloom filters. Section VIII evaluates the common-friend-estimation performance. Finally, Section IX concludes the paper.

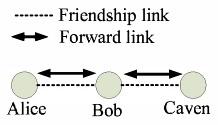


Fig. 1. Measuring common friends between two-hop neighbors Alice and Caven.

II. Related Work

Talash [8]calculates the common friends by exchanging two friend lists, which however, does not scale well with increasing friends; meanwhile, Talash also discloses sensitive personal information to unknown entities.

The Private Set Intersection (PSI) or private matching problem [9] aims to compute the intersection of two sets with privacy protection. Freedman et al. [9] introduce the two-party private matching problem and propose a seminal protocol for private matching. Later, several variants [10]–[12] of private matching protocols have been proposed in order to reduce the communication and computation overhead.

Song et al. [13] estimate diverse proximity metrics between users through a matrix factorization process, which needs a centralized storage of adjacency relations between users. However, the centralized assumption no longer holds for distributed online social networks. Besides, the dynamics of social networks requires the proximity metrics to be frequently updated that also increases the computation overhead of the factorization. Based on [13], Dong et al. [14] develop a secure and privacy-preserving dot product protocol to estimate social proximity in mobile social networks.

III. Background

A. Distributed Online Social Network

Let a **user** be an online entity that participates in the DOSN. A **friend** B of a user A is a user B that establishes the **social link** or **friendship link** with user A on the DOSN. A social link is a logical edge that connects two users A, B that means that users A, B can access the profiles and friend lists of each other. Furthermore, we call the set of users and the social links as a **social graph**. The number of **hops** between users A, B on the DOSN is the shortest path length between users A, B on the social graph. The **common friends** S_{AB} of two users A, B denote the subset of users that are friends to user A and B at the same time. Finally, each user A has a **friend list** S_A of a set of friends of user A and a **profile** that represents a set of personal records such as the personal status, or personal interests, etc.

B. Adversary Model

The goal of the privacy preservation is *not to disclose a user's friend list to users that are not his (or her) friends*. On the other hand, each user is able to request the friend list of his (or her) friend , since each user trusts his (or her) friends (denoted as the **one-hop trustiness**) and allows their friends to visit his (or her) friend list.

We assume that users are **semi-honest** [15]: users may be curious to learn the friend lists of users on other users, but do not claim fake friends about the friend lists of stored users. Particularly, each user follows the common-friend measurement protocol, but is able to eavesdrop on the communication links to or from him (or her).

C. Assumptions

Assume that each user is assigned a system-wide unique **identifier** when the user registers into the DOSN. The identifier is a randomized string (e.g., 160 bits by default) created by a cryptographic hash function like SHA-1. Due to the security of the hash functions, a curious user is unable to guess the identifier of a user due to the overwhelming computation overhead. For example, it is well known that finding two inputs with the same hashed strings is difficult for the SHA-1 hash function.

D. Bloom Filter

Given a set S of n elements, a standard Bloom filter $BF(S)$ represents S with a bit array I of the length m. Each bit $I[i]$ is initialized to zero for $i \in [1, m]$. When we **insert** an element y in to the set S, we use k independent hash functions $h_1, \ldots, h_k$ to map the element y into k random numbers within the interval $[1, m]$, such that each bit $I[h_i[y]]$ is set to 1 for $i \in [1, k]$. The Bloom filter supports the **query** "is $y \in S$?" by testing whether each bit $I[h_i[y]]$ is 1 for $i \in [1, k]$: if so, then y is assumed to be in the set, otherwise not. A Bloom filter may incur the **false positive** problem: if the k bits for the element z not in the set S have already been set by other elements in the set S, then the Bloom filter always returns that z is in the set S.

After we insert n elements into the set S, the false positive probability for an element not in S can be asymptotically computed by assuming that the hash functions are perfectly random [16]. Let p be the probability that a random bit in the Bloom filter is 0, then $p = (1 - 1/m)^{n \times k} \approx e^{-nk/m}$ [16]. Consequently, the false positive of the Bloom filter becomes:

$$FP = (1-p)^k \approx (1 - e^{-nk/m})^k \tag{1}$$

Furthermore, the optimal number of hash functions k that minimizes 1 with respect to a fixed n and m is given as [16]

$$k = \log 2 \times \left(\frac{m}{n}\right) \tag{2}$$

In fact, k should be a positive integer. So k is chosen as an integer slightly smaller than 2 for computational efficiency. The corresponding minimized false positive probability with respect to the optimal number of hash functions is [16]

$$0.6185^{m/n} \tag{3}$$

The standard Bloom filter does not support deleting stored items. Fortunately, it is not likely that an online user frequently deletes his (or her) friends. Therefore, we can use the standard Bloom filters to represent the friend lists.

IV. BCE

We present the common-friend estimation process in this section. First, we introduce the intersection of Bloom filters. Second, we present the common-friend estimation process using the common friends of a user pair.

A. Intersection of Bloom Filter

As discussed in the introduction section, we can use the identifiers of users to mitigate the dictionary attacks. Furthermore, using the Bloom filters to represent the set of friends' identifiers can hide the number of friends and scale much better than transmitting the identifier lists.

BCE estimates common friends of users based on the *i*ntersection of *B*loom *f*ilters (IBF). Suppose two users own the set S_A, S_B, and construct the Bloom filter $BF(A)$, $BF(B)$ with the same m and k, respectively. The IBF $BF(A) \cap BF(B)$ is defined as the bitwise AND result of $BF(A)$ and $BF(B)$:

$$BF(A) \cap BF(B) = BF(A)[i] \times BF(B)[i] \tag{4}$$

for $i \in [1, m]$. To estimate common friends, user A and B simply query items hashed into the IBF $BF(A) \cap BF(B)$ using S_A and S_B, respectively and take these items as the estimated common friends.

Since common friends must be hashed into the same locations in Bloom filters $BF(A)$ and $BF(B)$, the intersection of Bloom filters $BF(A) \cap BF(B)$ is similar with the Bloom filter $BF(A \cap B)$ that is constructed using the ground-truth set intersection $S_A \cap S_B$. As a result, the estimated common friends is quite close to the correct results. However, $BF(A) \cap BF(B)$ may differs a bit from $BF(A \cap B)$, since some bits in $BF(A) \cap BF(B)$ could be set by keys not in the set intersection.

B. Common Friends Estimation with One-hop Communication

To compute the intersection of Bloom filters, a naive approach is to exchange the Bloom filters between two two-hop users. However, disclosing a user's Bloom filter to non-friend users may *disclose partial friends of a user*. Suppose user C has two friends A and B. Users A and B own the set S_A, S_B, and construct the Bloom filter $BF(A)$, $BF(B)$, respectively. For example, a user A can query another user B's Bloom filters using a set of crawled identifiers over the DOSN to obtain a number of user B's friends.

First, to avoid the partial leakage of friends' information, each pair of friends only exchanges their own Bloom filters to each other. As a result, each user caches the Bloom filters of all his (or her) friends. As a result, user A and B only exchanges his (or her) Bloom filter with user C; while user C exchanges Bloom filters with user A and B.

Second, each user computes the IBF for each friend pair using cached Bloom filters. For example, user C computes the intersection of Bloom filters $BF(A) \cap BF(B)$ for A and B, and pushes $BF(A) \cap BF(B)$ to A and B, respectively. Then, user A and B can determine the set of common friends between each other without the need of knowing the other's Bloom filter. Fig 2 summarizes the above common-friend measurement process.

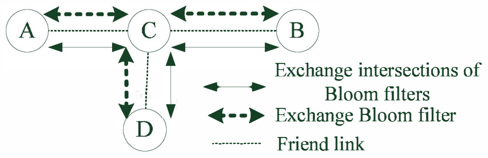


Fig. 2. Common-friend measurements.

V. Dynamic Friendship Representation

Based on the above discussions, we represent a friend list with a Bloom filter for scalability. However, for a static Bloom filter, its false positive rate keeps increasing with increasing friends, due to the dynamics of friend links between users.

To control the false positive rate of the Bloom filter despite of the friend dynamics on the DOSN, we adaptively tune the size m of the Bloom filters. On the other hand, we experimentally verify that a wide number of hash functions lead to similar false positive rates. As a result, we can fix the number k of hash functions to be constant for Bloom filters with variable lengths.

Suppose that two users A, B represents sets S_A, S_B with Bloom filters F_A, F_B independently. Let τ be the pre-specified false positive threshold. We know the minimal size of the Bloom filter m needs to be at least

$$m_{(n,k,\tau)} = \frac{-nk}{\ln\left(1 - \tau^{\frac{1}{k}}\right)} \tag{5}$$

in order to limit the false positive rate to be below the threshold τ. Therefore, we periodically set the size of the Bloom filter

As a result, we adaptively tune the size m of the Bloom filter as

- Let the default size of m be 1 KBytes.
- When the false positive FP of the Bloom filter exceeds τ, we increase the Bloom filter as $\left(m_{(n,k,\tau)} + 1024 * 8\right)$.
- We reconstruct the Bloom filter for the updated friend list with m and k.

Furthermore, we compute the hash functions with the SHA-1 cryptographic functions and set the hash value as the seeds. Algorithm 1 shows the hash process. As a result, each user immediately learns how to compute the hash values by knowing number of hash functions k of another Bloom filter.

VI. Computing the Intersection of two Variable-length Bloom Filters

To compute the intersection of two variable-length Bloom filters, we can map the larger Bloom filter to a smaller Bloom filter and compute the intersection using two small Bloom filters that are of the same length. However, if the size of two Bloom filters varies a lot, then mapping the large Bloom filter will incur a much higher false positive rate.

Algorithm 1 Computing the hash values for an element key.

```
1: function HASH($key, k, m$)
2:     $seed \leftarrow 0$.
3:     for $i = 1 \rightarrow k$ do
4:         $seed \leftarrow SHA1Hash(key, seed)$.
5:         $result[i] \leftarrow |seed \bmod m|$.
6: return $result$.
```

Therefore, we have to transform the small Bloom filter to a larger Bloom filter for accuracy. To that end, each user maintains a large Bloom filter $BF0$ in the memory with a length of S bytes(say 1 MBytes). Then, similar to the Buffalo [17], the user can flexibly create a new Bloom filter that of any length below 1 MBytes without the need of recreating the whole Bloom filter.

For two Bloom filters with size m_1 and m_2 ($m_1 > m_2$). Let the number of hash functions be k. Let

$$c = \lceil S/m_1 \rceil$$

Let $BF3$ be a new Bloom filter with m_1 bits and k hash functions. Then, the i-th bit in $BF3$ should be the module 2 sum of the bits at the i, $2i$, ..., ci -th bits in $BF0$.

VII. Probabilistic Privacy of the Bloom Filter

Assume that an attacker maintains a dictionary of identifiers crawled over the DOSN. Intuitively, the attackers can not know exactly whether an identifier is indeed in the set, since the Bloom filter is a probabilistic data structure with a false positive probability for a query. Since the false positives are statistical valid for any query, a user is always able to deny that he (or she) is a friend of another user due to the false positive.

Particularly, we first show that a Bloom filter provides some kind of probabilistic privacy, where each item can claim that it does not belong to the set. To do that, we show the **sensitivity** of a Bloom filter to a specific identifier. We construct two Bloom filters using two sets that differ in only one item. If these Bloom filters are quite similar to each other; then, an adversary can not distinguish whether a item is in the set.

Our probabilistic protection of the queried results is similar to the recent advances of the differential privacy [18], which adds noises to the query results. However, we argue that adding noises may be inappropriate for the Bloom filter. Since the Bloom filter is represented by a 0-1 string, replace 1 with other non-zero value is meaningless; while replacing 0 with non-zero values will significantly increase the false positives of the Bloom filter, which makes the Bloom filter less useful.

A. *Sensitivity for Intersection of Bloom Filter*

Instead of exchanging the Bloom filters, we release the intersection of Bloom filters to each other. As a result, we have to show the sensitivity of the intersection of Bloom filter with respect to a particular identifer. If the sensitivity is quite low, we can conclude that the intersection of Bloom filters provides high privacy protection, since adding or removing a specific identifer does not change the intersection of the Bloom filters. As a result, when an adversary node obtains the intersection of two Bloom filters, the node is not sure about whether an identifier is in the set or not.

Let $f_I : (S_3, S_4) \rightarrow IBF$ be a function that maps two sets S_3, S_4 to the intersection of two Bloom filters that are constructed by these two sets. Given two sets $S_3 = \{x_1, \ldots, x_n\}$, $S_3' = \{x_1, \ldots, x_n, x_{n+1}\}$ that differ in only one identifier x_{n+1}. We compute the L_1 difference of the mapping function f_I over two intersection of Bloom filter:

$$g\left(S_3, S'_3, S_4\right) \rightarrow \sum_{i=1}^{m} 1_{IBF(S_3,S_4)[i] \neq IBF(S'_3,S_4)[i]} \tag{6}$$

(1) **Upper bound**: $g\left(S_3, S'_3, S_4\right) = k$;

(2) **Lower bound**: $g\left(S_3, S'_3, S_4\right) = 0$;

(3) **Expectation**: Let the bit arrays of the IBF $BF(S_3) \cap BF(S_4)$, $BF(S_3') \cap BF(S_4)$ be I_3, I_4, respectively. Since the sets S_3 and S_3' differ in only one item, the only chance that the i-th bits in IBF $I_3[i]$ and $I_4[i]$ differ is that $I_3[i] = 0$ and $I_4[i] = 1$. Due to the independence of the false positives of the IBF $BF(S_3) \cap BF(S_4)$, $BF(S_3') \cap BF(S_4)$, the probability $P\left(I_3[i] = 0 \wedge I_4[i] = 1\right)$ that the i-th bit of $BF(S_3)$ is zero and that of $BF(S_4)$ is 1 can be written as:

$$\begin{aligned} & P\left(I_3[i] = 0 \wedge I_4[i] = 1\right) \\ & = P\left(I_3[i] = 0\right) \times P\left(I_4[i] = 1\right) \\ & = e^{-\frac{n_3 k}{m}} \left(1 - e^{-\frac{n_4 k}{m}}\right) \end{aligned}$$

Let Z_l be the indicator function for the event that $I_3[a_l] = 0$ and $I_4[a_l] = 1$ for the l-th bit out of k hash positions $\{a_1, \ldots, a_k\}$, where $a_l \in [1, m]$:

$$Z_l = \begin{cases} 1 & I_3\left[a_l\right] = 0 \wedge I_4\left[a_l\right] = 1 \\ 0 & else \end{cases}$$

Let Z be the random variable of the number of different bits in two IBFs:

$$Z = \sum_{l=1}^{k} Z_{a_l} \tag{7}$$

Due to the linearity of the expectation, we can write the expectation of the number of different bits as:

$$\begin{aligned} & E\left[Z\right] = E\left[\sum_{l=1}^{k} Z_{a_l}\right] \\ & = \sum_{l=1}^{k} E\left[Z_{a_l}\right] \\ & = \sum_{l=1}^{k} \left(1 \times P\left(I_3[a_l] = 0 \wedge I_4[a_l] = 1\right)\right) \\ & = \sum_{l=1}^{k} \left(1 \times e^{-\frac{n_3 k}{m}} \left(1 - e^{-\frac{n_4 k}{m}}\right)\right) \\ & = k \times e^{-\frac{n_3 k}{m}} \left(1 - e^{-\frac{n_4 k}{m}}\right) \end{aligned}$$

As a result, the expected number of different bits for two IBFs is $k \times e^{-\frac{n_3 k}{m}} \left(1 - e^{-\frac{n_4 k}{m}}\right)$. Fig 3 plots the expected number of different bits with increasing number of items. We can see that the expected number of different bits is less than one, which implies that two IBFs are identical in most cases. Furthermore, we can see that increasing the number of items in two Bloom filters also decreases the number of different bits in two Bloom filters.

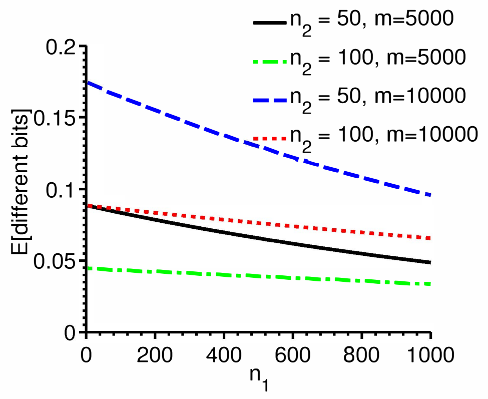


Fig. 3. Expected number of different bits for the IBF with increasing items n.

B. Summary

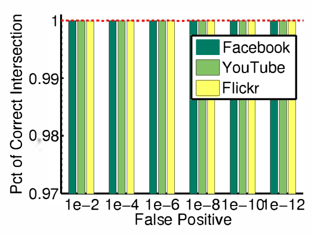
Intuitively, if a Bloom filter is the same as before after we insert an identifier into it, then any identifier can claim that it is not represented by that Bloom filter, since the Bloom filter is **insensitive** to an identifier. To do that, we compute the L_1 distance of two Bloom filters that are constructed by two sets that differ in only one item. We found that the number of different bits of two Bloom filters decreases quickly with increasing items, but is still bigger than 1 in most cases. Therefore, directly exchanging the Bloom filters may leak the information of a specific identifier.

On the other hand, we also compute the L_1 distance of two intersections of Bloom filters that differ in only one item. We found that the number of different bits is less than 0.2 in most cases and decreases quickly with increasing items. As a result, the intersection of Bloom filters is insensitive to a specific identifier, which implies that it preserves the privacy much better than exchanging the Bloom filters.

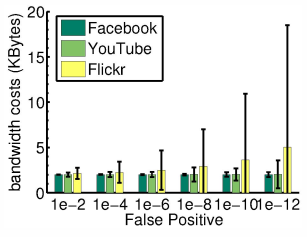
VIII. Evaluation

We represent each user with a 160-bit randomized string generated with a SHA-1 cryptographic hash function. The experiments are repeated in ten times and we report the mean results and the corresponding standard deviations. We analyze the characteristics of the friends on the real-world OSNs. We choose three representative social graphs that are all collected by the online social networks research group in the Max Planck Institute for Software Systems [19]–[21].

We compute the **Percentage of correct intersection** as the percentage of correct common-friend results by the Bloom filter based common-friend probing process as well as the bandwidth cost of transmitting the Bloom filter.



(a) Accuracy



(b) Bandwidth

Fig. 4. Accuracy and bandwidth costs by tuning the threshold of false positive threshold τ.

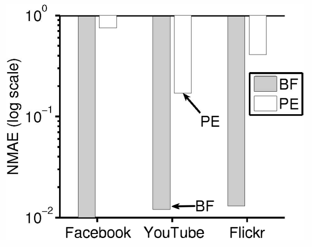


Fig. 5. Comparing the accuracy of different methods. The y-axis is plotted in log scale.

We first test whether there exists a suitable false positive threshold τ for the Bloom filter, such that the common-friend probing results are accurate and the probing bandwidth is also modest. Fig 4 show the dynamics of the accuracy and bandwidth costs by tuning the false positive threshold τ of the Bloom filters. We can see that most common-friend results are accurate as the threshold τ decreases. For example, when τ is no larger than 10^{-6}, more than 99% of all probing results are accurate, and further accuracy improvement is negligible. On the other hand, decreasing τ does not significantly increase the mean bandwidth costs, but increases the differences of probing bandwidth for different users, which causes imbalance probing bandwidth costs. As a result, we choose the false positive threshold τ to be 10^{-6} to trade off the bandwidth costs and the probing accuracy.

We compare our method (denoted as BF) with the matrix factorization based estimation method [13] (denoted as *PE*). We compute the Normalized Mean Absolute Error (NMAE) between the estimated number $\widehat{Y}_{ij}$ of common friends and Y_{ij}, the ground-truth number of common friends:

$$NMAE = \frac{\sum_{(i,j)} \left| Y_{ij} - \widehat{Y}_{ij} \right|}{\sum_{(i,j)} Y_{ij}} \qquad (8)$$

Smaller NMAE values correspond to higher prediction accuracy. From Figure 5, we see that the Bloom filter outperforms the proximity embedding method in orders of magnitudes, which is consistent with Figure 4. The proximity embedding method however, incurs much higher errors since the number of common friends is not strictly low-dimension.

IX. Conclusion

We propose the problem of estimating common friends in distributed social networks in order to recommend new friends without disclosing personal information. To that end, we propose a Bloom filter based common-friends prediction scheme that protects the privacy of users' friends and balances the communication and computation loads. Since exchanging the Bloom filters may partially leak the privacy of some friends, each user computes the intersection of Bloom filters for two-hop users that are both their friends. We provide a detailed analysis for the probabilistic privacy protection. Finally, simulation results on real-world data sets confirm the accuracy and efficiency of the common-friend estimation.

References

[1] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: Feasibility of Transitive Cooperation for Privacy on a Decentralized Social Network," in *Proc. of IEEE WOWMOM'09*, pp. 1–6.

[2] L. M. Aiello and G. Ruffo, "LotusNet: Tunable Privacy for Distributed Online Social Network Services," *Comput. Commun.*, vol. 35, pp. 75–88, 2012.

[3] T. Xu, Y. Chen, J. Zhao, and X. Fu, "Cuckoo: Towards Decentralized, Socio-aware Online Microblogging Services and Data Measurements," in *Proc. of HotPlanet '10*, pp. 4:1–4:6.

[4] Diaspora, "Diaspora Alpha," https://joindiaspora.com/, September 2011.

[5] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "PeerSoN: P2P Social Networking: Early Experiences and Insights," in *Proc. of SNS '09*, 2009, pp. 46–52.

[6] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. A. Li, D. Liu, and A. Varshavsky, "Vis-à-Vis: Privacy-preserving Online Social Networking via Virtual Individual Servers," in *Proc. of COMSNETS'11*, 2011, pp. 1–10.

[7] D. Liben-Nowell and J. Kleinberg, "The Link Prediction Problem for Social Networks," in *Proc. of CIKM '03*, pp. 556–559.

[8] R. Dhekane and B. Vibber, "Talash: Friend Finding In Federated Social Networks," in *Proc. of LDOW2011*.

[9] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient Private Matching and Set Intersection," in *Proc. of EUROCRYPT 2004*, pp. 1–19.

[10] E. D. Cristofaro, P. Gasti, and G. Tsudik, "Fast and Private Computation of Set Intersection Cardinality," Cryptology ePrint Archive, Report 2011/141, 2011, http://eprint.iacr.org/.

[11] L. Kissner and D. Song, "Privacy-Preserving Set Operations," in *Proc. of CRYPTO 2005*, pp. 241–257.

[12] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung, "Efficient Robust Private Set Intersection," in *Applied Cryptography and Network Security*, 2009, vol. 5536, pp. 125–142.

[13] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu, "Scalable Proximity Estimation and Link Prediction in Online Social Networks," in *Proc. of IMC '09*, pp. 322–335.

[14] W. Dong, V. Dave, L. Qiu, and Y. Zhang, "Secure Friend Discovery in Mobile Social Networks," in *Proc. of IEEE INFOCOM'11*.

[15] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2: Basic Applications.

[16] A. Z. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, 2003.

[17] M. Yu, A. Fabrikant, and J. Rexford, "BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations," in *CoNEXT*, 2009, pp. 313–324.

[18] C. Dwork, "A Firm Foundation for Private Data Analysis," *Commun. ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[19] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the Evolution of User Interaction in Facebook," in *Proc. of WOSN '09*, pp. 37–42.

[20] M. Cha, A. Mislove, and K. P. Gummadi, "A Measurement-Driven Analysis of Information Propagation in the Flickr Social Network," in *Proc. of WWW '09*, pp. 721–730.

[21] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *Proc. of IMC'07*.