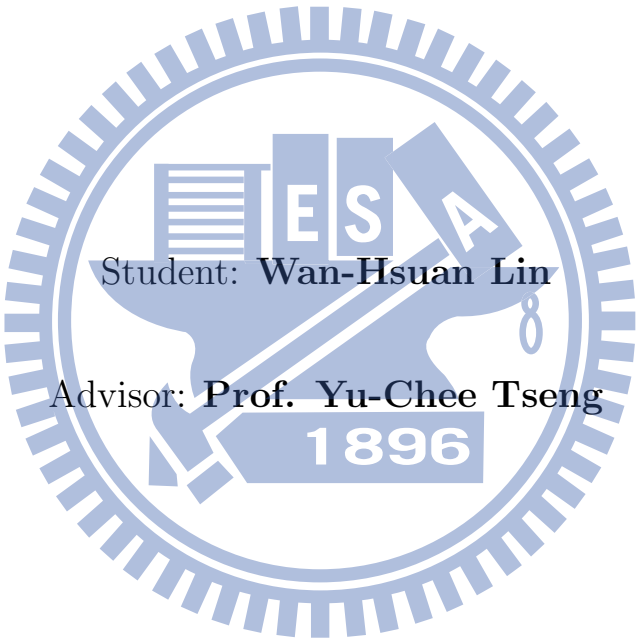
On Common Profile Matching among Multiparty Users in Mobile D2D Social Networks



Student: **Wan-Hsuan Lin**

Advisor: **Prof. Yu-Chee Tseng**

Department of Computer Science

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan 300, R.O.C.

應用於行動D2D社群網路之
多方用户資訊匹配方法

學生: 林宛萱　　指導教授: 曾煜棋 教授

國立交通大學資訊工程與科學研究所 碩士班

摘要

近年來隨著智慧型手機的盛行，社群網路的發展已經相當純熟， 因此本論文研究主要應用在社群網路上，並且特別針對使用者行動裝置已經是在近距離的情況， 我們稱作行動D2D社群網路。有關行動D2D社群網路其中一項重要應用，就是共同資訊的匹配， 所謂的共同資訊的匹配，是指使用者都各自攜帶行動裝置，相遇在某個空間(例如:宴會廳)， 並且這群使用者都感興趣彼此之間共同具有的特性，藉由裝置間短距離通訊(稱作D2D)來傳送使用者的資訊並且完成匹配。 舉例來說對於一群相互陌生的使用者，可能會想知道彼此間共同的嗜好、朋友或者是過去曾經都去過某些國家， 對於學生而言，可能是想知道共同修過的課程。 本論文應用在行動D2D社群網路，並假設每位使用者的裝置都可以和其他裝置直接互相通訊下 ，制定了三種有關資訊匹配的問題，分別是全共有問題、β-共有問題、前γ熱門的問題， 第一個問題主要是延伸之前的相關研究，而後面兩個問題是此篇論文新制定的。 我們提出的方法會根據基本布隆過濾器及疊代布隆過濾器，利用布隆過濾器的特性解決本篇資訊匹配的三個問題， 最後利用實作來驗證我們提出方法達到將低通訊成本的目標。

關鍵字: 布隆過濾器、D2D 通訊、社群網路、資訊匹配、智慧手機應用軟體

On Common Profile Matching among Multiparty Users in Mobile D2D Social Networks

Student: Wan-Hsuan Lin Advisor: Prof. Yu-Chee Tseng

Department of Computer Science
National Chiao Tung University

ABSTRACT

Recently, *mobile social networks (MSNs)* have been widely discussed due to the rapid growth of smart mobile devices. This work focuses on *mobile D2D social networks (MDSNs)*, where users in an MSN are physical neighbors. An important social application of MDSNs is *common profile matching (CPM)*, which refers to the scenario where a group of smartphone users meet in a small region (such as a ball room) and these users are interested in identifying the common attributes among them from their personal profiles efficiently via short-range (such as D2D) communications. For example, a group of strangers may want to find common hobbies, friends, or countries they visited before, and a group of students may want to know the common courses they have ever taken. Assuming that users in an MDSN form a fully connected network, we formulate three versions, namely *all*-common, β-common, and top-γ-popular, of the CPM problem. The first problem is an extension of an earlier work, while the latter two problems are newly defined. We present solutions based on the *basic* and the *iterative Bloom filters*. Evaluation results show that our mechanisms are quite communication-efficient.

Keywords: Bloom filter, D2D communication, mobile social network, profile matching, smartphone apps.

誌　　謝

首先由衷感謝曾煜棋老師對我耐心的指導和鼓勵，我非常佩服曾老師的EQ以及專業，他是個親和力很強的老師，在我研究的道路上時常遇到困難，老師除了正面的鼓勵我，並會指導我另外一個可行的方向，提醒我不要太過沮喪，因此將近兩年下來，才能順利完成碩論，在曾老師的身上也學習到很多有關研究該有的態度和方法，此外，也感謝曾老師提供的研究學習環境，不需要擔心實驗的器材設備，讓我在最後階段實作時，也不用爲了設備煩惱。亦感謝所有口試委員陳文村教授、周百祥教授、陳伶志教授以及陳烈武教授，口試當天所給予的建議與鼓勵，讓我的碩論可以更加完整也更加豐富。

另外我也要感謝HSCC實驗室這個大家庭，學長姊們總是很願意提供他們的專業知識，並且盡可能給予我學習上相關的幫助及建議，尤其特別感謝同組的陳彥安學長、巫芳璟學姊、胡淑學姊，這將近兩年的照顧和協助，而其中主要帶領我的陳彥安學長，總願意私底下額外再花費時間，與我討論碩論中遭遇的問題，從題目的制定到方法最後是驗證的種種過程，非常謝謝他的這之中不厭其煩的教導與討論，從口頭報告到論文寫作，多虧有學長的帶領和鼓勵，我才能有這篇碩論成果。也感謝同屆的同學們，讓我的研究生涯裡，除了能一起面對困難也能一起分享所有喜悅。

最後，我要感謝我親愛的家人，特別是我父母，讓我在學習的道路上不用煩惱其他事情，能夠勇往直前，因爲有他們的支持，才有完成這篇碩論的我。

林宛萱　於

國立交通大學資訊科學與工程研究所碩士班

中華民國一百零四年一月

Contents

List of Figures

Chapter 1

Introduction

Mobile social networks (MSNs) have become increasingly popular due to the explosive growth of smart mobile devices. Smartphones and pads usually have multiple communication interfaces, such as Bluetooth, WiFi, GSM, WCDMA, and LTE, which allow them to communicate with other remote devices via infrastructures or nearby devices via direct or D2D (device-to-device) communications. D2D communications have many promising applications, such as content dissemination, advertisement, broadcasting, location-aware services, gaming, and social interaction. In particular, when smart devices communicate with others via D2D communications for socialization purposes, we call the underlying network a *mobile D2D social netwo43rk (MDSN)*.

Lubricating social interaction is one of the most important MDSN applications. When people meet new friends or attend social events, they may be eager to find out some common topics or backgrounds for initiating conversations with others. For example, a group of strangers may want to find common hobbies, friends, or countries they visited before to chat, and a group of students may want to know the common courses they have taken to discuss. Observing the rapid growth of smart mobile devices, it is possible to improve such social interactions or social experiences with the assistance of MDSN. *Common profile matching (CPM)* refers to the need of finding some common attributes of a group of smartphone users in a small region under an MDSN. Recently, many MDSN applications for improving social interactions have been proposed. References [16, 29, 34] propose privacy-preserving mechanisms for finding a best matched user among a group of users or the common profile of two persons. Reference [23] presents an intuitive device pairing method based on measuring the time difference of two sound events. Reference [27] exploits layered information publishing and directional localization techniques for lubricating social interactions. Reference [32] demonstrates a handshake matching mechanism for authenticating information exchange between two users. Reference

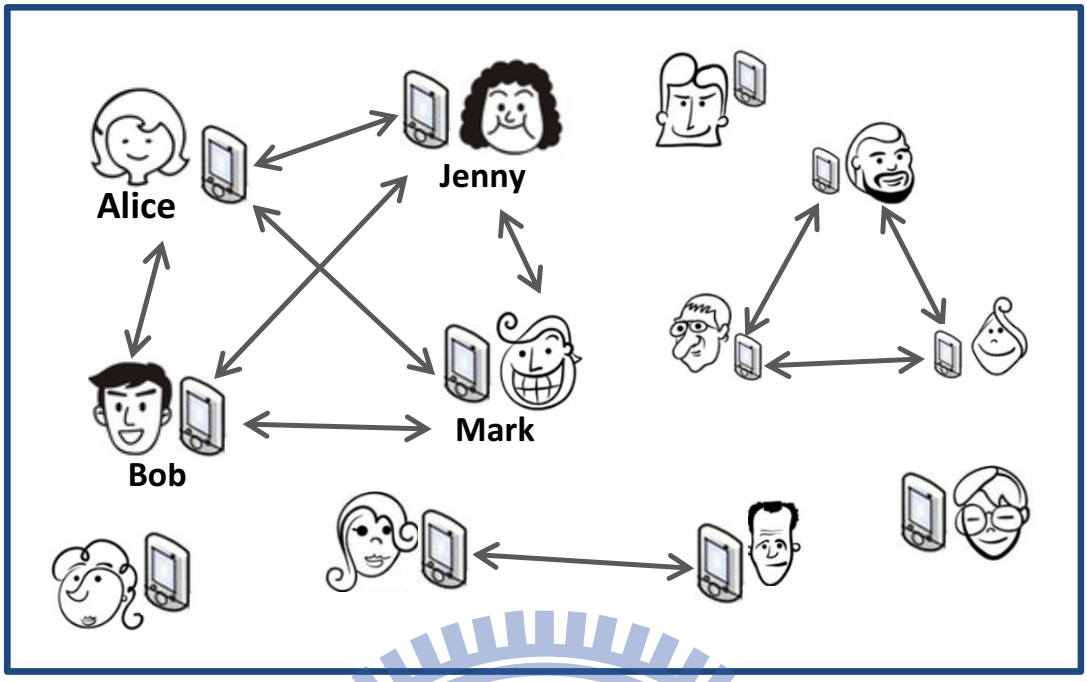


Figure 1.1: Our CPM application scenario.

[33] leverages the Bluetooth Service Discovery Protocol to publish user-defined profiles. However, these applications mainly focus on the social interaction between two devices in proximity. Social interactions among multiparty users, i.e., many-to-many social interaction, in an MDSN have not been well addressed yet. A multiparty private matching scheme is proposed in [15] for database systems, but it does not consider the message cost incurred during matching.

In this work, we consider the CPM issue in an MDSN where users are willing to cooperate to find some common attributes among them without privacy violation. Note that user profiles are available in many social networks, such as Facebook, Twitter, and LinkedIn, for this purpose. Fig. 1.1 shows the application scenario. Each user carries a smart mobile device with D2D communication capability. A profile is stored in each mobile device, which contains a list of attribute profiles of the user. For example, a profile may look like {interest = {NBA, swimming}, course-taken = {linear algebra, operating system, C language}, country-visited = {Taiwan, USA, Slovak, Austria}}. Here, "interest", "course-taken", and "country-visited" are called *attribute profiles*, and "NBA", "swimming", etc. are called *attribute items*. Users are in proximity, but may form multiple logical groups. Users in the same logical group agree to execute the matching process, for example, via enabling an app on their smart phones. (We comment that *neighbor discovery* [2, 7, 13], and group formation need to be done prior to the CPM process, but we will omit discussing these parts.) We formulate three CPM-related

problems: *all-common*, *β-common*, and *top-γ-popular* problems, where β and γ are integers. The first problem is an extension of the earlier work [33], while the latter two are newly defined. The first problem is to find the common attribute items of all users. The second problem is to find the common attribute items of at least β users, while the third problem is to find the top-γ attribute items shared by most users. We present communication-efficient solutions to these problems based on *basic* and *iterative Bloom filters*. We exploit Bloom filter since it is a space-efficient data structure for comparing the items owned by two parties without publishing their entire attribute profiles. In addition, it can provide privacy against eavesdroppers due to its property of one-way hashing.

The main contributions of this work are as follows. First, we have formulated three CPM problems that aim at lubricating social interactions among a group of physically close users via D2D communications. Second, compared to existing MDSN applications, which are designed for one-to-one communications, our solutions are for multiparty users and work in a more efficient way. Third, we have prototyped these applications via Android apps. Implementation experiences are presented, which do verify their feasibility.

The rest of this paper is organized as follows. Related our works are reviewed in Section 2. Preliminaries are given in Section 3. Section 4 presents our solutions. Section 5 shows our prototyping results and performance evaluation. Finally, Section 6 concludes this paper.

Chapter 2

Related Works

First, We survey related works in MSNs. Then, we review some profile matching works.

Reference [14] discusses the applications, architectures, and protocol designs of MSNs. A MSN usually needs centralized servers to exchange and share information among mobile devices. Reference [36] presents the design pattern of a web-based MSN service. All information exchanges need to go through web servers. In [11], a middleware is proposed to provide a common platform for rapidly developing MSN applications. The platform enables capturing, managing, and sharing the social states of physical communities and offloads the computation of MSN applications. Location-based services for sharing comments, photos, videos, or activities are addressed in [10, 20, 21]. Among MSN applications, we are more interested in MDSN, which allows mobile devices in proximity to communicate directly with each other without relying on server. Such D2D communication technologies are actually quite mature (such as Wi-Fi Direct [30] and Wi-Fi hotspot [31]). In [24], a middleware is developed for maintaining an MDSN over ad-hoc networks by store-carry-and-forward techniques. Reference [22] presents a query/response matching service in a mobile ad hoc Bluetooth/Wi-Fi network. Reference [28] infers the shared interests between users with RF-based devices by analyzing patterns of physical proximity of these people. Reference [33] facilitates social networking in physical proximity by automatically suggesting common topics between two users in close vicinity.

For profile matching,references [16, 17, 29, 34] focus on the privacy-preserving issue. In [16], fully distributed protocols are proposed such that an initiating user can find the best matched person from a group of users. Reference [17] proposes comparison-based profile matching protocols, which maintain anonymity during matching. However, these private matching mechanisms do not consider the incurred message cost and computation overhead. Reference [29] proposes a matchmaking mechanism such that users can exchange attributes while others cannot identify the contents. Reference [34] further considers fine-grained private matching. Reference

[8] addresses social serendipity without concerning the privacy issue. The above works do not consider the message and computation overheads. Reference [33] does address the message cost, but as the works reviewed above, it cannot handle multi-party CPM. In this sense, our *all*-common, β-common, and, top-β-popular CPM problems are more general.

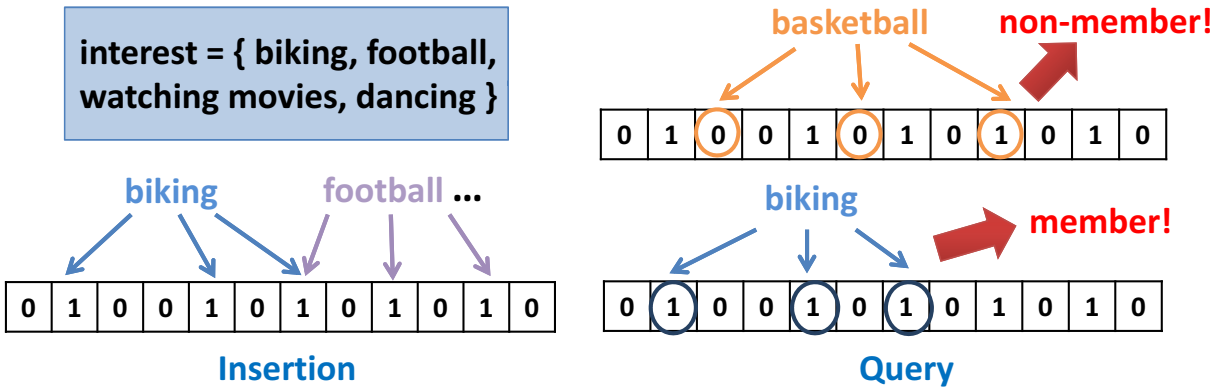


Figure 3.1: An example of insertion and query to a Bloom filter.

Chapter 3

Preliminaries

We briefly introduce the basics of Bloom filter [3], which is a space-efficient probabilistic data structure for storing membership data. Given a set S, a Bloom filter allows two basic operations: *insertion* and *query*. Initially, the Bloom filter is an array of m bits with all 0s . To insert an element $s \in S$ into this Bloom filter, we use z independent hash functions each taking s as the input and mapping to an integer in $[1, m]$. Let $h()$ be any hash function. Then the $h(s)$-th entry of the m-bit array is set to 1. Repeating this process for all z hash functions, the insertion of s into the Bloom filter is done. If we repeat the above procedure for all members of S, the insertion of S is done.

To query whether any s' is a member of S, we compute z hash values from the same z hash functions using s' as the input. For each hash function $h()$, we check if the $h(s')$-th bit of the array is 0. If so, s' is not a member of S. Otherwise, s' is very likely to be a member of S. Fig. 3.1 illustrates an example of inserting an "interest" attribute profile to a Bloom filter and two query examples.

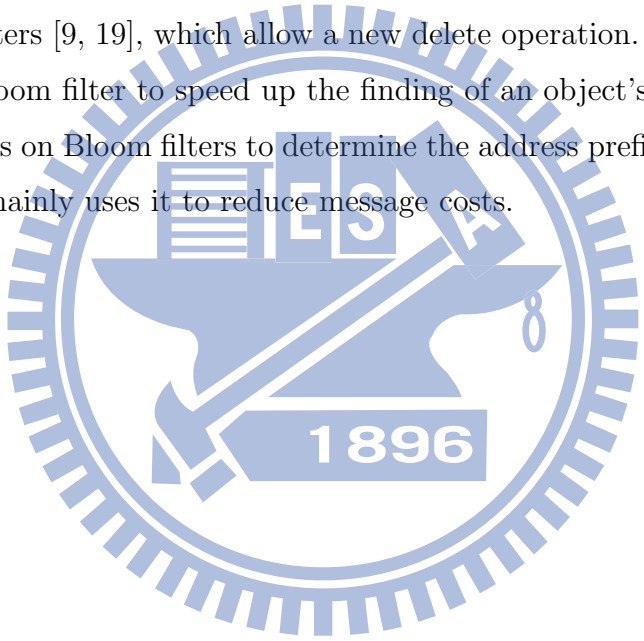
Since Bloom filter is a probabilistic data structure, it may regard a non-member as a member

by mistake, or known as *false positive*. The probability of a false positive, also called *false positive rate*, is

$$f = (1 - (1 - \frac{1}{m})^{zn})^z \approx (1 - e^{-zn/m})^z \quad ,$$

where m is size of the Bloom filter, z is the number of hash functions, and $n = |S|$. It represents the probability that the corresponding z hashed bits are all 1s while s' is not a member of S [5]. Therefore, given predefined n, z, and, f values, we can choose a proper array size m. Note that however there is no false negative.

Bloom filter provides a tradeoff between space complexity and query accuracy. For publish-subscribe systems, [12, 35] use it to reduce memory and bandwidth costs when storing and propagating users' information. It is also possible to extend the original Bloom filter to *counting* and *weighted* Bloom filters [9, 19], which allow a new delete operation. Reference [18] provides a name service with Bloom filter to speed up the finding of an object's location. Reference [6] performs parallel queries on Bloom filters to determine the address prefix membership in sorted prefix sets. Our work mainly uses it to reduce message costs.



Chapter 4

Common Profile Matching in an MDSN

4.1 System Model

In an MDSN, we consider one user group U and one *attribute profile* of these users. (An attribute profile could be the hobbies, the countries visited, the courses taken, or the celebrities followed by users. Solving this problem for multiple user groups and attribute profiles can be done by repeating this for each user group and attribute profile.) Let $U = \{u_1, u_2, \ldots, u_q\}$ and the attribute profile of user $u_i \in U$ be P_i, $i = 1..q$. Each element in P_i is called an *attribute item*. Let $n_i = |P_i|$. The universal set of the attribute items is denoted by P and we assume that P is known by all users.

We propose three versions of the CPM problem.

- *all*-common: The goal is to find the intersection of all users' attribute profiles, i.e., $C_{\text{all}} = \bigcap_{i=1..q} P_i$.
- β-common: The goal is to find the set of all attribute items such that each attribute item is in at least β users' attribute profiles. For any $a \in P$, we define a membership function $\eta(a, P_i)$ such that $\eta(a, P_i) = 1$ if $a \in P_i$ and $\eta(a, P_i) = 0$ otherwise. The β-common set is defined as $C_{\beta\text{-com}} = \{a | \Sigma_{i=1..q} \eta(a, P_i) \geq \beta\}$.
- top-γ-popular: The goal is to find the set of top-γ hottest attribute items that are shared by all users' profiles. The top-γ set is defined as $C_{\text{top-}\gamma} = \{a |$ the value $\Sigma_{i=1..q} \eta(a, P_i)$ of a is ranked top-$\gamma\}$.

4.2 Basic Bloom Filter Solution

We present a basic Bloom filter solution for solving the CPM problems. Assuming that users in U form a fully-connected network, our solution first requests each user u_i, $i = 1..q$, to insert her attribute profile P_i into a Bloom filter and sends the m-bit array to all other users in U. On receiving any u_j's m-bit array, $i \neq j$, u_i tries to recover u_j's attribute profile P_j. From these collected attribute profiles (which possibly contain false-positive elements), u_i then tries to calculate C_{all}, $C_{\beta\text{-com}}$, and $C_{\text{top-}\gamma}$.

Definition 1. *Given any attribute profile P_i, define $BF(P_i)$ to be the m-bit array obtained by inserting each attribute item in P_i into the Bloom filter. Given any set $S \subseteq P$, define $Qry(S, BF(P_i))$ to be the set containing each element $a \in S$ which returns a positive answer when querying the existence of a in the Bloom filter $BF(P_i)$, i.e., $Qry(S, BF(P_i)) = \{a|\ (a \in S) \wedge (a \in BF(P_i))\}$. (Here, $a \in BF(P_i)$ means that for each hashed value of a, the corresponding bit in $BF(P_i)$ is 1.)*

The basic solution is detailed as follows. These steps are executed by each user $u_i, i = 1..q$, concurrently. (Refer to Fig. 4.1 for illustration.)

1. User u_i computes its Bloom filter array $BF(P_i)$ and broadcasts $BF(P_i)$ to all other users.
2. User u_i collects the Bloom filter arrays $BF(P_j)$ of all other users $u_j, j \neq i$.
3. For each $BF(P_j)$ collected from u_j, u_i computes the set $P'_j = Qry(P, BF(P_j))$, which is an estimation of P_j.
4. Finally, u_i uses its own P_i and all other $(q-1)$ estimated $P'_j, j \neq i$, to compute *all*-common set C_{all}, β-common set $C_{\beta\text{-com}}$, and top-γ-popular set $C_{\text{top-}\gamma}$ (this involves trivial set and ranking operations, so we omit the details).

Although this solution is very simple, it may have the following potential drawbacks. First, since the false positive rate is related to the value of m discussed in Section 3, the communication cost could be large when m is large. Second, in step 3, the query process needs to exhaustively check each attribute item in P. The processing complexity $\mathcal{O}(q \cdot |P|)$ is high when the universal set P or the user number q is large. Note that if someone queries by the universal set, the privacy of users may still be bleached. To resolve this problem, users can determine a shared secret key [4, 26]. Then we can concatenate the secret key to the attribute items when computing hash values. Without the secret key, it is hard to recover the original attribute items.

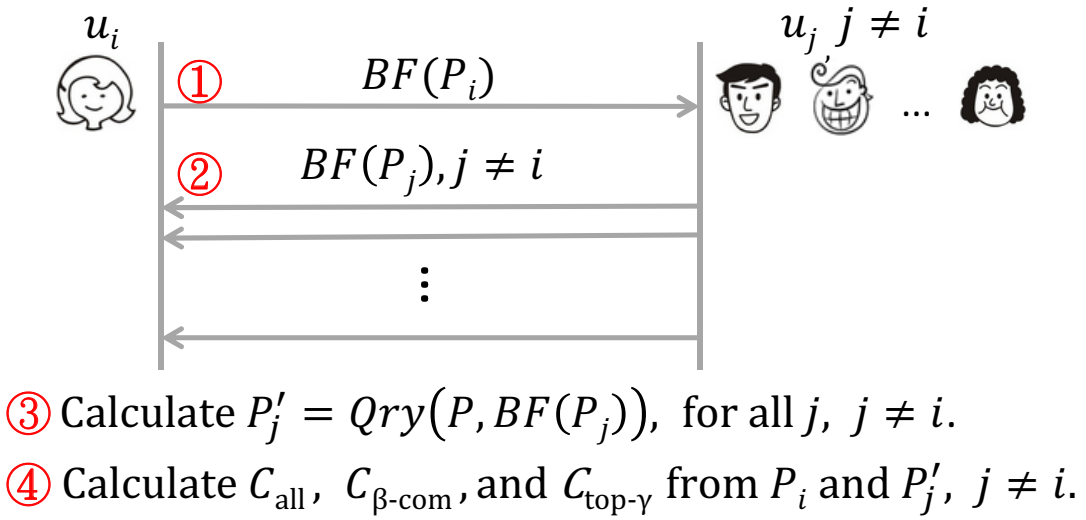


Figure 4.1: Workflow of the basic Bloom filter solution.

4.3 Iterative Bloom Filter Solution

This solution is based on the iterative Bloom filter (IBF) [33]. The goal is to further reduce the message cost by using smaller Bloom filters iteratively. Recall that the size of a Bloom filter should be set according to the expected number of inserted items and the expected false positive rate. We can use a smaller Bloom filter (thus with a higher false positive rate) in the first iteration to reduce message cost. Since some items have been filtered out, in the subsequent iterations, we can still use smaller Bloom filters while controlling the false positive rate. Therefore, IBF may keep the same false positive rate as the basic solution with smaller message cost. In [33], a 2-iteration IBF has been shown to be quite efficient. Hence, we adopt the same approach.

The following steps are executed by each user $u_i, i = 1..q$, concurrently. Note that the value of m is not necessarily the same as that used in the basic Bloom filter solution. (Refer to Fig. 4.2 for illustration.)

1. User u_i computes its Bloom filter array $BF(P_i)$ and broadcasts $BF(P_i)$ to all other users.
2. User u_i collects the Bloom filter arrays $BF(P_j)$ of all other users $u_j, j \neq i$.
3. For each $BF(P_j)$ collected from u_j, u_i computes the set $P_{i,j} = Qry(P_i, BF(P_j))$, which is an estimation of $P_i \cap P_j$. (It contains the set of attribute items that appear in P_i as well as in the Bloom filter $BF(P_j)$. Note that the query cost of $Qry(P_i, BF(P_j))$ could be much less than that of $Qry(P, BF(P_j))$ when P is large. Also note that $P_{i,j}$ and $P_{j,i}$ may not be the same.)

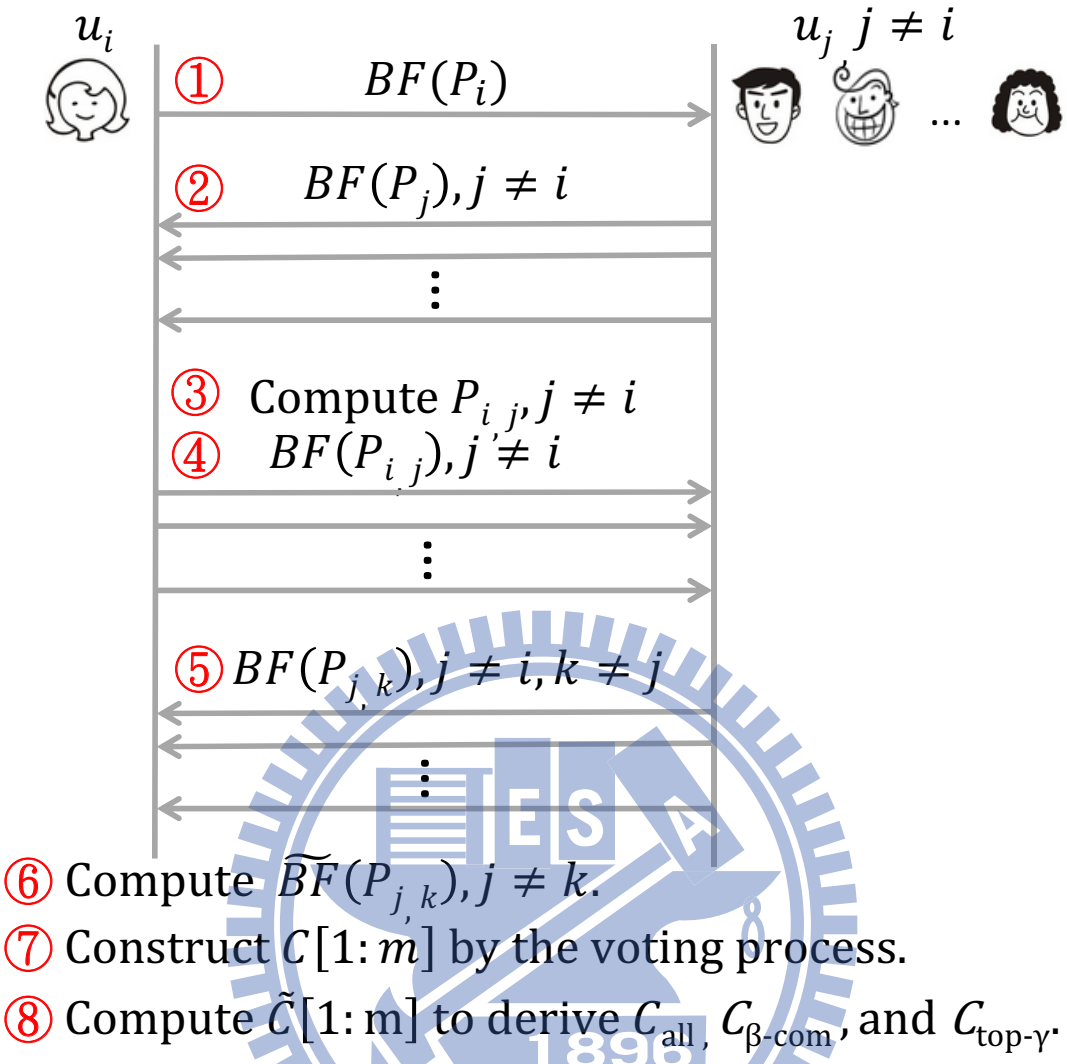


Figure 4.2: Workflow of 2-iteration IBF solution.

4. In step 3, u_i already has the sets $P_{i,j}$ for all $j \neq i$. Then, u_i computes $(q-1)$ Bloom filter arrays $BF(P_{i,j}), j \neq i$, and broadcasts these arrays to all other users.

5. User u_i collects $(q-1)$ Bloom filter arrays $BF(P_{j,k})$ from each u_j, $j \neq i, k \neq j$. (There are totally $(q-1)^2$ arrays received.)

6. Combining the $(q-1)$ arrays in step 3 and the $(q-1)^2$ arrays in step 5, u_i now has $q(q-1)$ arrays, namely $BF(P_{s,t})$ for all $s \neq t$. Then u_i computes the vector

$$\widetilde{BF}(P_{s,t}) = BF(P_{s,t}) \wedge BF(P_{t,s}),$$

where $\wedge$ is the bit-wise logical AND operator. It is not hard to see that $\widetilde{BF}(P_{s,t}) = \widetilde{BF}(P_{t,s})$.

7. Next, u_i uses a voting process to construct an integer array $C[1:m]$. The value of each $C[d]$ is initially 0, $1 \leq d \leq m$. Intuitively, in the voting process, each user can cast one (and only one) vote to $C[d]$ if any hashed value of any attribute item is d. To calculate $C[d]$, we use a tentative set T_d, which is set to $\emptyset$ initially. Then we check the d-th bit of each $\widetilde{BF}(P_{j,k})$. There are three cases:

   - If $\widetilde{BF}(P_{j,k})[d] = 0$, do nothing.
   - If $\widetilde{BF}(P_{j,k})[d] = 1$ and only one of u_j and u_k is in T_d, increase $C[d]$ by 1 and include the one of u_j and u_k which is not in T_d into T_d.
   - If $\widetilde{BF}(P_{j,k})[d] = 1$ and none of u_j and u_k is in T_d, increase $C[d]$ by 2 and include both u_j and u_k into T_d.

   Note that the above process should be repeated for each $\widetilde{BF}(P_{j,k})$. This process avoids querying each $\widetilde{BF}(P_{j,k})$ by the universal set P to reduce computation cost. Compared to the basic Bloom filter solution, this step may slightly increase the false positive rate of the final results. To summarize, this voting process helps quickly estimate the number of votes that $C[d]$ receives.

8. Using array $C[1:m]$, u_i derives its answers to the three CPM problems as follows.

   - *all*-common: We convert $C[1:m]$ to an array $\widetilde{C}[1:m]$ such that $\widetilde{C}[d] = 1$ if $C[d] = q$ (i.e., q votes) and $\widetilde{C}[d] = 0$ otherwise. The answer of C_{all} is $Qry(P, \widetilde{C})$.
   - β-common: We convert $C[1:m]$ to an array $\widetilde{C}[1:m]$ such that $\widetilde{C}[d] = 1$ if $C[d] \geq \beta$ (i.e., β votes) and $\widetilde{C}[d] = 0$ otherwise. The answer of $C_{\beta\text{-com}}$ is $Qry(P, \widetilde{C})$.
   - top-γ-popular: The answer of $C_{\text{top-}\gamma}$ can be obtained by repeating the β-common problem by setting $\beta = q$ and gradually collecting the attribute items by decreasing the value of β by 1 each time, until γ attribute items are collected or β is equal to 1.

Chapter 5

Prototyping Results and Performance Evaluation

5.1 Application Prototyping

We implement an MDSN application for CPM on Android phones as shown in Fig. 5.6. The development of an Android application is based on Android software development kit (SDK) [1] and is written in Java programming language. The goal is for users to know the common visited countries among nearby users. The MDSN is implemented by WiFi hotspot. In the future, we may use WiFi Direct. Fig. 5.3 shows the screenshots of our Android app. In this example, there are four users. At first, they press the "Start" button to join the MDSN (Fig. 5.3(a)). One of them is in charge of being a network router by enabling the WiFi hotspot. Users can choose their attribute items from the universal set (Fig. 5.3(b)). The user profiles of these users are presented in Fig. 5.2. When ready, users can press the "MATCH" button to run the CPM procedure (Fig. 5.3(c)). After CPM is completed, they can check the results (Fig. 5.3(d), (e), and (f)). The software architecture presented in Fig. 5.1 is composed of the modules of user interface, profile manager, transmission, and CPM algorithm. Below, We explain each component in details.

- *User Interface*: We provide simple and clear interfaces for user to manipulate our app Fig. 5.3. The first job of this component is for user to launch the network connection (i.e., establishing or joining an MDSN) by touching the button "Start" without any other manual settings. In addition, touching the button "Matching" launch the CPM procedure. Second, it provides interfaces for user to configure and check his personal profiles. The user can look over matching results of each matching problem respectively as shown in (Fig. 5.3(d), (e), and (f)).

- *Profile Manager*: This component can store attribute profiles, such as "interest", "course-

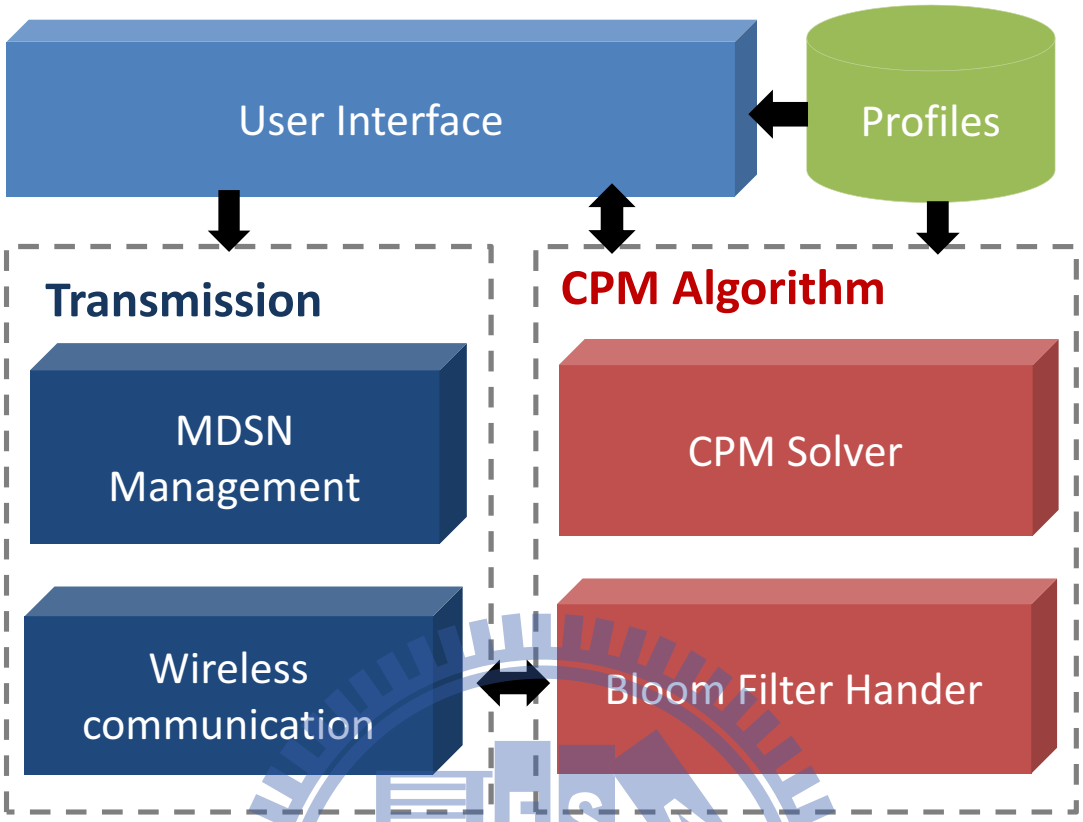


Figure 5.1: The architecture of our MDSN application for CPM.

Alice	{ Austria, America, Bhutan, Brazil, China, Chile, Canada, Egypt, France, Germany, Guinea, Iceland, Japan, Kenya, Korea South, Nepal, Italy, Norway, Philippines, Portugal, Singapore, Switzerland, Taiwan, Thailand, Turkey }
Bob	{ America, Australia, Belgium, Brazil, Canada, Djibouti, France, Germany, Greece, Iceland, Italy, Japan, Finland, Korea South, Lebanon, Libya, Micronesia, New Zealand, Portugal, Singapore, Spain, Taiwan, Turkey, Vietnam }
Mark	{ America, Brunei, Canada, Chile, China, Germany, Greece, Czech, Egypt, France, Italy, Iraq, Japan, Korea South, New Zealand, Netherlands, Norway, Spain, Switzerland, Taiwan, Thailand, Turkey, Ukraine, Vanuatu }
Jenny	{ America, Australia, Belgium, Czech, Ecuador, Estonia, Finland, France, Gabon, Greece, Hungary, Japan, Korea South, New Zealand, Netherlands, Norway, Philippines, Spain, Taiwan, Thailand, Turkey, Qatar, Romania}

Figure 5.2: The personal profile of each user.

taken" and "country-visited", provided by a user. Each attribute profile contains attribute items like {interest = singing, dancing, swimming, running, and so on}. The user-configured profiles are the inputs of CPM algorithms. In this application prototyping, we choose the country-visited (e.g., the countries that a user had been to before.) for demonstration. Hence, profile manager maintains the handset user's country-visited profile and the countries around the world as the universal attribute items of a country-visited

profile.

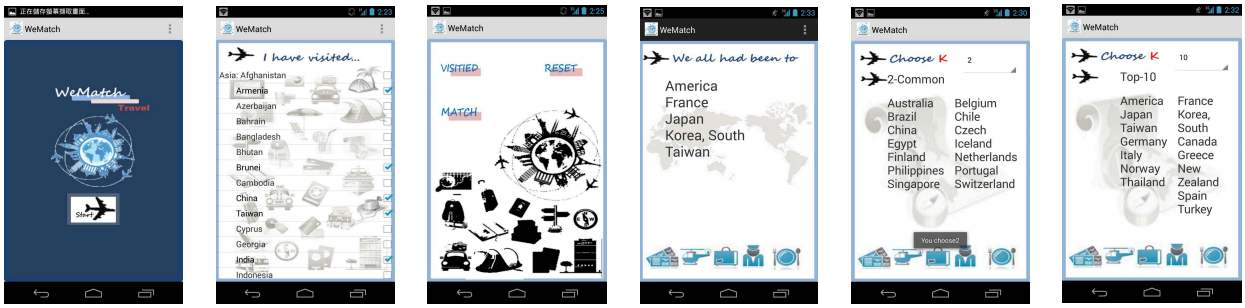
- *Transmission*: As for the D2D communication, we exploit Wi-Fi hotspot on Android to realize the MDSNs. It is because this WiFi hotspot feature is backward compatible to older Android phones. Thus, WiFi is the interface of wireless communication function in this module. For controlling WiFi of Android phone, WifiManager and WifiConfiguration in Android SDK are the fundamental API. However, there is no API to directly configure WiFi hotspot in Android SDK. We use a Java reflection technique to control and configure the functionalities of WiFi hotspot. The device opening the WiFi hotspot will act as an access point (AP) while the other devices can connect to it as a client.

  When executing this application, the transmission module first enables WiFi to search an access point with a predefined SSID. If this AP exists, the smartphone automatically connects to it for joining the MDSN. Otherwise, the smartphone enables the Wi-Fi hotspot with the predefined SSID. We call this WiFi hotspot enabled smartphone as AP device and it is responsible for exchanging the necessary data among connected devices. Therefore, the function *MDSN Management* provides for AP device to manage the members of the MDSN. All the other smartphones in this MDSN communicate with each other through the AP device.

- *CPM Algorithm*: This component contains two functions, *Bloom Filter (BF) Handler*, *CPM Solver*. BF Handler leverages the Bloom filter to insert the user-configured profiles. As discussed in Section 4, there are two versions of BF Handler, i.e., basic Bloom filter and IBF. Since the performance of IBF is better than basic BF, we adopt IBF in this application. After finishing exchanges of Bloom filters, CPM solver executes the voting process presented in Section 4.3 for efficiently computing the results of the *all*-common, β-common, and top-β-popular CPM problems.

5.2 Evaluation Results

We evaluate the proposed solutions by two metrics, execution time and message cost, on our application prototype. The evaluation testbed is six commercial Android smartphones (Google Nexus 4) as shown in Fig. 5.6. The parameters in the experiments are as follows. We assume that $z = 7$ and use *MD5* [25] to generate hash values. In the basic Bloom filter solution, the expected false positive rate f is set to 0.001%. In the iterative Bloom filter solution (2



(a) (b) (c) (d) (e) (f)

Figure 5.3: Screenshots of the prototyped Android app: (a) Touch the "Start" to participate in an MDSN. (b) Select the countries the user visited before. (c) Touch the "MATCH" to start the CPM procedure with nearby users. (d) The answer to the *all*-common problem. (e) The answer to the β-common problem. (f) The answer to the top-γ-popular problem.

iterations), we respectively set $f = 10\%$ and $f = 0.01\%$ for the first and the second iterations. The total false positive rate is also 0.001% as that of the basic Bloom filter solution. In addition, we also use the exact strings of attribute items as the baseline solution. We set the average string length of attribute items to 10.

Message size is the major factor that affects the computation and communication performance. Recall that we determine the Bloom filter size m by considering the potential number of inserted items n , the expected false positive rate f , and the number of hash functions z. Here, we study the performance issue of the *all*-common problem by varying n and f with different numbers of users.

1. Varying n: We vary n from 200 to 1000 in each user's attribute profile. We observe the execution time, which contains computation and communication time during matching. Fig. 5.4 shows that basic BF and the IBF solutions perform much better than the baseline solution when n is large. This is because Bloom filter reduces message size with its hashing mechanism. Also, the result shows that Bloom filter is computation-efficient since the execution time increases slightly when n increases. Fig. 5.4 shows that the gap between basic BF and IBF shrinks when the number of users increases. This is because the messages exchanged during the IBF procedure increase as the number of users increases. Fig. 5.5 shows the message costs of the entire network incurred by these solutions. As expected, IBF significantly outperforms baseline and basic BF solutions in message costs. This is because IBF uses smaller Bloom filter in each iteration.

2. Varying f: We vary f for IBF in its second iteration from 0.001% to 10%. Note that the value of f in the first iteration is fixed. (We omit the performance of basic BF because it

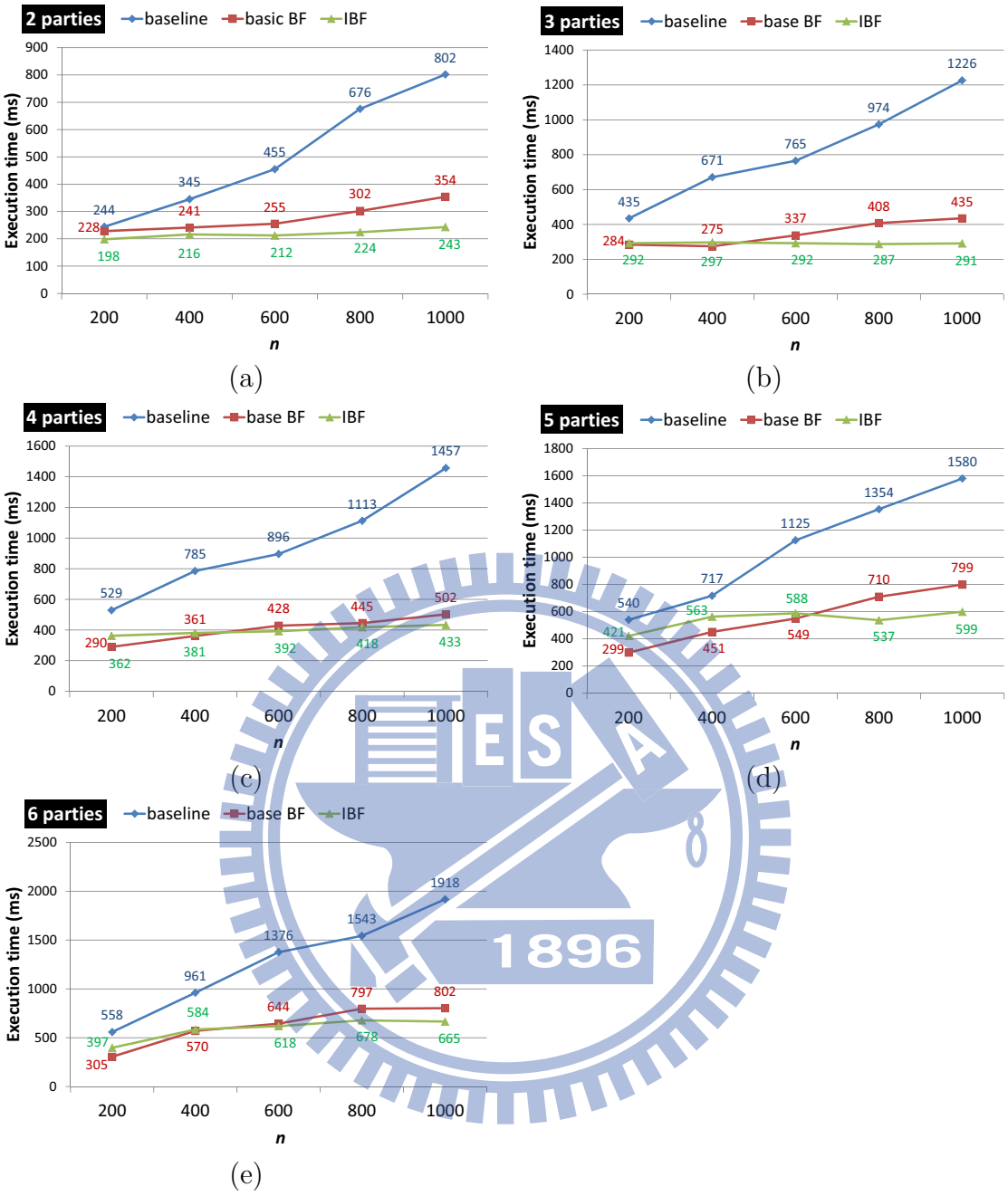


Figure 5.4: Comparison of execution time by increasing the number of inserted items to a Bloom filter with (a) two parties, (b) three parties, (c) four parties, (d) five parties and (e) six parties.

is similar to that of IBF.) In this experiment, we set $n = 1000$. However, in the second iteration of IBF, the number of attribute items to be inserted is the intersection of two sets. We assume here that the expected size of the intersected set is 50. We then use these values and f to choose the Bloom filter sizes in the first and the second iterations of IBF. Fig. 5.7(a) shows that the execution time only decreases slightly when f increases. The reason is that the intersected set size of the second iteration (50) is much smaller

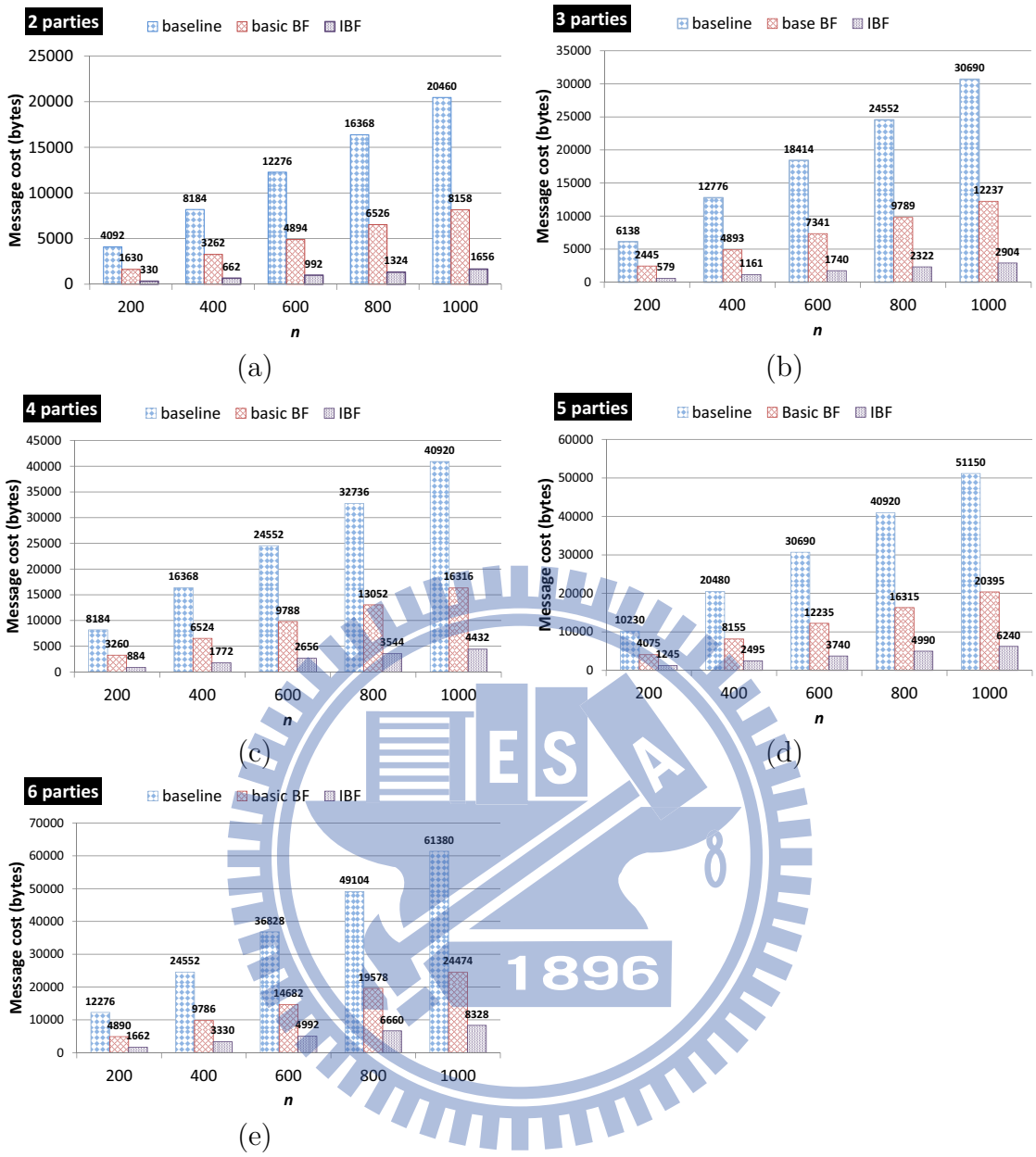


Figure 5.5: Comparison of message cost by increasing the number of inserted items to a Bloom filter with (a) two parties, (b) three parties, (c) four parties, (d) five parties and (e) six parties.

than that of the first iteration (1000). Because m is proportional to n if f is fixed, the reduced message size is small when varying f in IBF's second iteration. Fig. 5.7(b) shows the message costs of the entire network in different cases. To sum up, varying f of IBF's second iteration will not significantly affect the execution time.

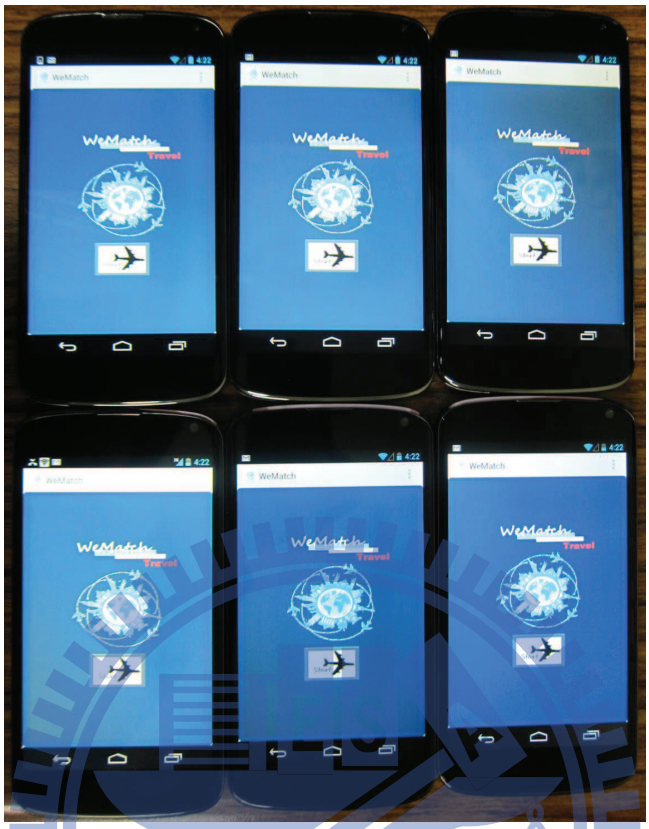
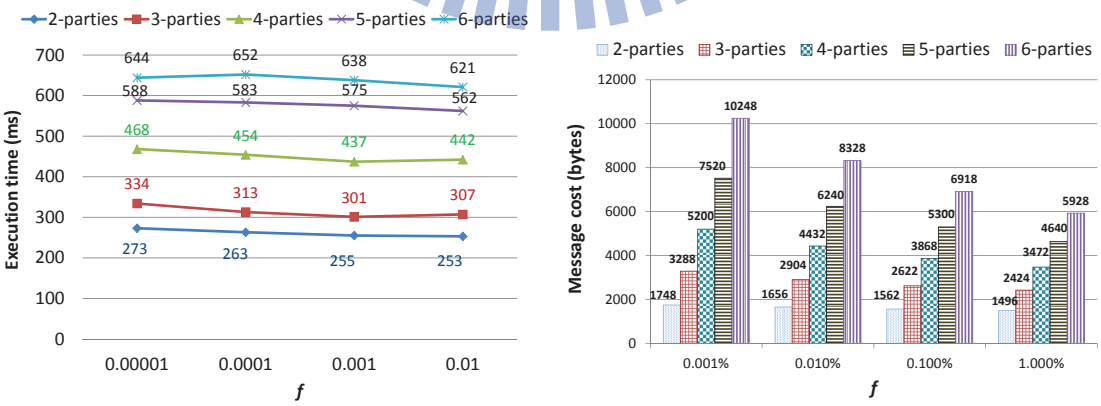


Figure 5.6: Application prototyping on Android phones and evaluation testbed.



(a) (b)

Figure 5.7: Comparison of (a) execution time and (b) message cost by increasing the expected false positive rate of the IBF.

Chapter 6

Conclusions

In this work, we have presented CPM in an MDSN, which is to identify common attributes among physical neighbors via D2D communications. We consider three CPM problems: *all*-common, β-common, and top-γ-popular. We propose basic and iterative Bloom filter-based solutions to these problems. We have conducted some evaluations on commercial smartphones to prove that the proposed solutions are communication-efficient. The basic Bloom filter and IBF solutions perform well in execution time when the number of attribute items in each profile is increased. However, the IBF solution incurs lower message costs. We have also demonstrated a prototype on Android smartphones. The application verifies that the proposed solutions are feasible on off-the-shelf smartphones.

Bibliography

[1] Android sdk. `http://developer.android.com/sdk/index.html`.

[2] M. Bakht, M. Trower, and R. H. Kravets. Searchlight: won't you be my neighbor? In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2012.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group diffie-hellman key exchange. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2001.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[6] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *Proc. of ACM Special Interest Group on Data Communication (SIGCOMM)*, 2003.

[7] P. Dutta and D. Culler. Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[8] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 4(2):28–34, 2005.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *ACM Trans. on Networking*, 8(3):281–293, 2000.

[10] Foursquare. `http://foursquare.com/`.

[11] A. Gupta, A. Kalra, D. Boston, and C. Borcea. Mobisoc: A middleware for mobile social computing applications. *Mobile Networks and Applications*, 14(1):35–52, 2009.

[12] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander. Llipsin: Line speed publish/subscribe inter-networking. In *Proc. of ACM Special Interest Group on Data Communication (SIGCOMM)*, 2009.

[13] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar. U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol. In *Proc. of International Conference on Information Processing in Sensor Networks (IPSN)*, 2010.

[14] N. Kayastha, D. Niyato, P. Wang, and E. Hossain. Applications, architectures, and protocol design issues for mobile social networks: A survey. *Proceedings of the IEEE*, 99(12):2130–2158, 2011.

[15] P. K. Y. Lai, S.-M. Yiu, K. P. Chow, and C. F. Chong. An efficient bloom filter based solution for multiparty private matching. In *Proc. of International Conference on Security and Management (SAM)*, 2006.

[16] M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2011.

[17] X. Liang, X. Li, K. Zhang, R. Lu, X. Lin, and X. S. Shen. Fully anonymous profile matching in mobile social networks. In *IEEE Journal on Selected Areas in Communications*, 2012.

[18] M. C. Little, S. K. Shrivastava, and N. A. Speirs. Using bloom filters to speed-up name lookup in distributed systems. *The Computer Journal*, 45(6):645–652, 2002.

[19] S. Liu, L. Kang, L. Chen, and L. Ni. Distributed incomplete pattern matching via a novel weighted bloom filter. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[20] Loopt. `http://www.loopt.com`.

[21] Mobiluck. `http://www.mobiluck.com`.

[22] M. Motani, V. Srinivasan, and P. S. Nuggehalli. Peoplenet: Engineering A wireless virtual social network. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2005.

[23] C. Peng, G. Shen, Y. Zhang, and S. Lu. Point&Connect: Intention-based device pairing for mobile phone users. In *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2009.

[24] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: Middleware for mobile social networking. In *Proc. of ACM Workshop on Online Social Networks (WOSN)*, 2009.

[25] R. Rivest. The MD5 message-digest algorithm. In *IETF RFC (1321)*, 2003.

[26] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to group communication. In *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 1996.

[27] J. Teng, B. Zhang, X. Li, X. Bai, and D. Xuan. E-shadow: Lubricating social interaction using mobile phones. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[28] M. Terry, E. D. Mynatt, K. Ryall, and D. Leigh. Social Net: Using patterns of physical proximity over time to infer shared interests. In *Proc. of ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2002.

[29] Y. Wang, T.-T. Zhang, H.-Z. Li, L.-P. He, and J. Peng. Efficient privacy preserving matchmaking for mobile social networking against malicious users. In *Proc. of IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2012.

[30] Wi-fi peer-to-peer (P2P) specification. `https://www.wi-fi.org/knowledge-center/published-specifications`.

[31] Android 2.2 platform highlights - portable hotspot. `http://developer.android.com/sdk/android-2.2-highlights.html`.

[32] F.-J. Wu, F.-I. Chu, and Y.-C. Tseng. Cyber-physical handshake. In *Proc. of ACM Special Interest Group on Data Communication (SIGCOMM)*, 2011.

[33] Z. Yang, B. Zhang, J. Dai, A. C. Champion, D. Xuan, and D. Li. E-SmallTalker: A distributed mobile system for social networking in physical proximity. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[34] R. Zhang, Y. Zhang, J. Sun, and G. Yan. Fine-grained private matching for proximity-based mobile social networking. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[35] Y. Zhao and J. Wu. B-sub: A practical bloom-filter-based publish-subscribe system for human networks. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[36] H. Zhong, L. Bi, Z. Feng, and N. Li. Research on the design method of mobile social network services. In *Proc. of International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII)*, 2008.