Mobility-Assisted Energy-Aware User Contact Detection in Mobile Social Networks

Wenjie Hu*, Guohong Cao*, Srikanth V. Krishanamurthy†, and Prasant Mohapatra‡
*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA
†Department of Computer Science and Engineering, University of California, Riverside, CA
‡Department of Computer Science ,University of California, Davis, CA
*{wwh5068, gcao}@cse.psu.edu, †krish@cs.ucr.edu, ‡prasant@cs.ucdavis.edu

***Abstract*—Many practical problems in mobile social networks such as routing, community detection, and social behavior analysis, rely on accurate user contact detection. The frequently used method for detecting user contact is through Bluetooth on smartphones. However, Bluetooth scans consume lots of power. Although increasing the scan duty cycle can reduce the power consumption, it also reduces the accuracy of contact detection. In this paper, we address this problem based on the observation that user contact changes (i.e., starts and ends of user contacts) are mainly caused by user movement. Since most smartphones have accelerometers, we can use them to detect user movement with much less energy and then start Bluetooth scans to detect user contacts. By conducting experiments on smartphones, we discover three relationships between user movement and user contact changes. According to these relationships, we propose a Mobility-Assisted User Contact detection algorithm (MAUC), which triggers Bluetooth scans only when user movements have a high possibility to cause contact changes. Moreover, we propose energy aware MAUC (E-MAUC) to further reduce energy consumption during Bluetooth discovery, while keeping the same detection accuracy as MAUC. Via trace driven simulations, we show that MAUC can reduce the number of Bluetooth scans by half while maintaining similar contact detection rates compared to existing algorithms, and E-MAUC can further reduce the energy consumption by 45% compared to MAUC.**

I. Introduction

Detecting users within the communication range (i.e., contact detection) is essential for many practical problems in mobile social networks such as designing routing protocols [22, 25], identifying community structures [14, 5, 10], analyzing social behaviors [18], etc. With the proliferation of smartphones and various embedded sensors, detecting user contacts using Bluetooth becomes a common solution [12, 18]. However, to detect user contacts with Bluetooth, an extremely power consuming process called Bluetooth *scan* must be performed. Frequent scans can obtain accurate results but will drain the phone battery quickly. In contrast, infrequent scans save power but may miss lots of user contacts.

There have been solutions on achieving a balance between accuracy and energy. The basic idea is to predict the next user contact time and adaptively adjust the Bluetooth scan duty cycle [20, 21, 18]. Although these solutions can save power by reducing the number of scans, they also reduce the accuracy of contact detection since many user contacts can not be detected due to the long duty cycle or prediction errors [18, 20].

The goal of this paper is to accurately detect user contacts with low power. Our solution is based on the observation that user contact changes (i.e., starts and ends of user contacts) are mainly caused by user movement. For example, two users not in contact meet each other only because one (or both) of them is moving toward the other. Similarly, two users within communication range lose contact because one (or both) of them moves away. Since most smartphones have accelerometers which consume much less energy, we can use them to detect user movement and then only detect user contacts during user movements.

Although using movement detection to trigger user contact detection can significantly reduce the energy consumption, there are two challenges: *(1) For moving users, how to adjust the Bluetooth scan duty cycle to reduce the number of scans and maintain detection accuracy? (2) How should users collaborate, so that static users can also detect user contact changes caused by moving neighbors?* To answer these questions, we collect Bluetooth scan traces and user movement traces from a small testbed consisting of 20 users. Based on the relationships between user contact changes and user movements observed from the traces, we propose a Mobility-Assisted User Contact detection algorithm (**MAUC**), to detect user contacts accurately. MAUC uses the accelerometer in the smartphone to detect user movement. Moving users only trigger Bluetooth scans when their movements have a high possibility of causing contact changes. Also, moving users will inform static users at proper time when they pass by. As a result, MAUC can reduce the number of Bluetooth scans while maintaining high detection accuracy.

For a Bluetooth device to be detected through a Bluetooth scan, it must stay in the Bluetooth discovery (discoverable) mode, which also consumes lots of power. To address this problem, we propose energy-aware MAUC (E-MAUC) to reduce the amount of time that a Bluetooth device has to stay in the discovery mode. Our detailed contributions are

This work was supported in part by the US National Science Foundation (NSF) under grant number CNS-1218597, and by Network Science CTA under grant W911NF-09-2-0053.

as follows.

- We collect and analyze user moving and contact traces and identify three relationships between user contact changes and user movement.
- We propose MAUC, an energy efficient algorithm to detect user contacts. It can improve the detection accuracy, while reducing the number of Bluetooth scans.
- To further save energy, we propose energy-aware MAUC to adaptively adjust the amount of time that the Bluetooth interface has to stay in the discovery mode.

The remainder of this paper is organized as follows. Section II gives an overview of our work. Section III analyzes the relationship between user contact changes and user movement. Section IV and Section V present the details of MAUC and E-MAUC. Section VI evaluates the performance of our solutions based on the collected traces. Section VII reviews related work and Section VIII concludes the paper.

II. Overview

A. Bluetooth Interface

For Android smartphones, the Bluetooth interface can work in two modes: *on* and *discovery*. A phone can only be found by others if it works in the discovery mode. Also, a phone needs to execute a Bluetooth *scan* process explicitly in order to find others. The power consumption of the smartphone is different when the Bluetooth interface is set to different modes. To measure the power consumption, we use Agilent E3631A Power Supply to provide the current with constant voltage (3.7V) to the smartphone instead of using the battery [23, 24]. We connect the Agilent E3631A to our laptop through the “NI GPIB-USB-HS” cable and program it by LabVIEW, in order to capture the current of the smartphone every 0.25 second.

Fig. 1 shows the power consumption of the smartphone when the Bluetooth interface is operating in different modes. As can be seen, keeping Bluetooth *on* costs very little power, while keeping it in *discovery* costs much more. The scan process is very long (around 12 seconds) and consumes lots of power. More details are summarized in Tab. I, where all the measurements are done for 10 times when the phone's screen is turned off. The power consumption is averaged over one minute when the phone is in various modes, except for Bluetooth scan, in which the power consumption is averaged during the scan process. Also, the value without star contains the phone's total power consumption (including the idle mode), while the value with star are measured as extra power in addition to that of the Bluetooth discovery mode.

The Bluetooth communication on Android is based on RFCOMM, which provides a simple and reliable data stream to users, like TCP. RFCOMM requires the two communication nodes to work as client and server to establish a connection. The server side first creates a service with a

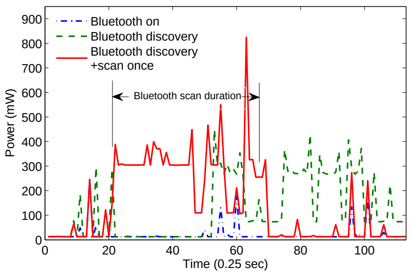


Figure 1. Power consumption of the smartphone when the Bluetooth interface is operating in different modes

Table I
POWER CONSUMPTION (MW)

Status	Mean	Std. dev
Idle	23.43	2.92
Bluetooth on	25.23	1.36
Bluetooth Discovery	140.10	21.56
Bluetooth Dis.+Server	141.06	41.42
Bluetooth Scan (12s)	307.59*	32.29
Accelerometer	1.78*	0.86

* means extra power consumption in addition to that of the Bluetooth discovery mode.

specified unique id `UUID`. Then the client can obtain the server's MAC address through Bluetooth scan, and connect to the server's MAC address using the same `UUID`.

B. Δ-detection Rate

Two users within the Bluetooth communication range can detect each other by Bluetooth scan. To measure the accuracy of a contact detection algorithm, the commonly used metric is *detection rate* ($\mathcal{D}$), which is defined as the number of successfully detected contacts divided by the total number of contacts. However, even if different detection algorithms have the same detection rate, their accuracy may vary.

Fig. 2 shows an example. The real contact duration between two users are shown in the solid shaded rectangle area. With Method-1, the contact is detected through Bluetooth scans, and the detected contact duration is shown in the upper dashed rectangle area. Similarly, Method-2 also detects the user contact. Although both methods have the same detection rate $\mathcal{D} = 1$, method-2 has higher detection accuracy since it detects the start and the end of the contact more accurately. That is, the detected contact duration of Method-2 (shown in the dashed rectangle) is closer to the real contact duration (shown in the solid rectangle) than that of Method-1. Thus, a new metric is needed to differentiate these two cases.

To overcome the limitation of detection rate, we introduce Δ-*detection* rate, which measures how many *contact changes* (i.e., the start and the end of the contact) are detected. Considering a contact change event e happens at t_e, and it is detected at t. If $|t - t_e| \leq \Delta$, e is successfully

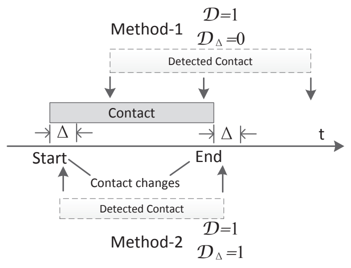


Figure 2. Using Δ-detection rate to measure the detection accuracy. Each vertical arrow represents a Bluetooth scan at that time.

detected. Then, *Δ-detection rate* ($\mathcal{D}_\Delta$) is defined as the number of successfully detected contact changes divided by the total number of contact changes.

As shown in Fig. 2, there are two contact changes, marked as Start and End. Method-1 detects neither of the contact changes, and hence its $\mathcal{D}_\Delta = 0$. Method-2 detects both of them and hence its $\mathcal{D}_\Delta = 1$. Therefore Method-2 is more accurate than Method-1.

C. Basic Idea

In this paper, we aim to accurately detect user contacts with less energy. More generally, we aim to improve the detection rate and Δ-detection rate, while reducing the number of Bluetooth scans. Fig. 3 illustrates the basic idea. Traditional algorithms do not consider user movement. They assume that the Bluetooth interface always stays in the discovery mode, and use Bluetooth scans to detect neighbors periodically. Although there are solutions to adjust the scan duty cycle to save power based on some prediction models, the detection accuracy is also reduced due to the long duty cycle and high prediction errors.

In MAUC, based on the observation that user contact changes are mainly caused by user movement, accelerometers are introduced to detect user movement. Then Bluetooth scans are only triggered when a user movement is detected, either by the users themselves or their neighbors, as shown in Fig. 3. Also, we propose techniques to achieve a balance between reducing the number of Bluetooth scans and improving the detection accuracy.

Since staying in Bluetooth discovery also consumes lots of power as shown in Tab. I, the smartphone should work in the discovery mode only when others are scanning. However, this is not supported by the original Android system. To address this problem, we modify Android so that the discovery mode can be turned on/off adaptively. Then, we design energy aware MAUC (E-MAUC), which only sets Bluetooth in discovery mode for a short period of time while still satisfying the discovery requirement of MAUC (see Fig. 3). Thus, E-MAUC can keep the same detection accuracy while saving more energy. More details of E-MAUC will be described in Section V.

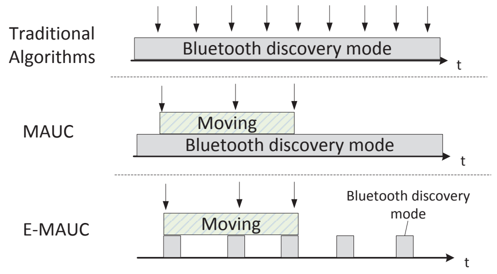


Figure 3. Comparison of traditional contact detection algorithms and the proposed solutions

III. Trace Analysis

In this section, we introduce our traces and identify the relationships between user movement and contact changes.

A. Trace Collection

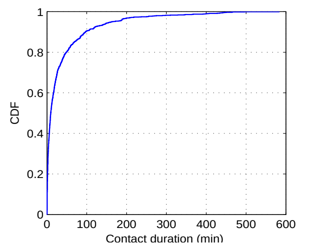
To collect the user contact and moving traces, we distributed Samsung Nexus S phones to 20 students in the CSE and IST Departments at the Pennsylvania State University and conducted experiments for 10 days during the period from 8am to 8pm. The phones run a background service which initiates a Bluetooth scan every 30 seconds and records the detected neighbors. The first time a neighbor appears (disappears) is considered as the start (end) of a contact with that neighbor. The logged user contacts are used as the ground truth to determine the accuracy of various user contact detection algorithms.

The accelerometer is sampled at 50Hz, and the mean, standard deviation and peak values of the three axes are recorded within a window of 5 seconds. We classify users' mobility status as moving or static within that window using a decision tree algorithm that will be described in Section IV-B. The duration over which a user's mobility status is classified as moving is recorded as a *moving period*.

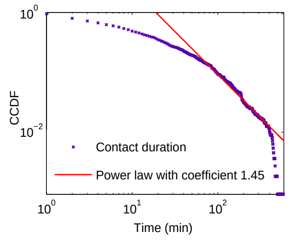
B. User Contact Duration

The distribution of user contact duration affects the Bluetooth scan duty cycle. First, the scan interval should be smaller than most contact durations. Otherwise, most short user contacts will be missed. Second, this distribution also affects how to measure the accuracy of the detection algorithms. For example, in Fig. 2, two algorithms both detect a user contact but with different contact duration, then it is necessary to use Δ-detection rate to differentiate them, especially when most contact durations are short.

The cumulative distribution function (CDF) of the user contact duration is shown in Fig. 4(a). As can be seen, 50%, 80% and 90% of user contact duration is less than 10 minutes, 50 minutes, and 100 minutes, respectively. The average user contact duration is 36 minutes, which is close to that in the MIT Reality trace [2]. As 15% of the contacts are shorter than 1 minute in our trace, we set Δ to 30 seconds. Additionally, the $CCDF$ (complementary CDF) of user

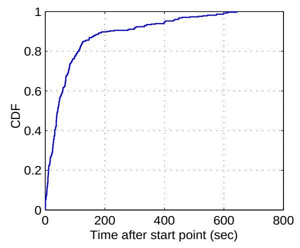


(a) CDF of user contact duration

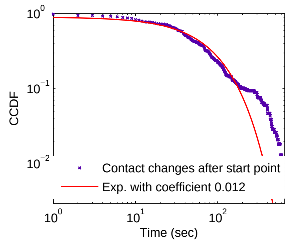


(b) User contact duration pattern

Figure 4. User contact duration distribution

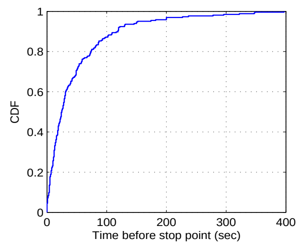


(a) CDF of the time between contact changes and start points

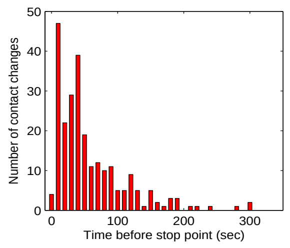


(b) CCDF of the time between contact changes and start points

Figure 5. Relationship between contact changes and start points

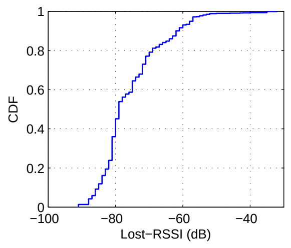


(a) CDF of the time between contact changes and end-points

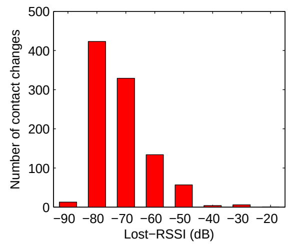


(b) Pattern of the time between contact changes and end-points

Figure 6. Relationship between contact changes and end-points



(a) CDF of lost-RSSI



(b) Pattern of lost-RSSI

Figure 7. Relationship between communication range and lost-RSSI

contact duration in Fig. 4(b) (log-log scale) shows that user contact duration follows power law distribution within the range [60 min, 400 min] with a scaling index of 1.45, which is consistent with the result in [3].

C. Relationships Between Contact Changes and User Movement

Intuitively, a contact change event usually happens at the start or end of a moving period. For example, when a student moves from one classroom to another, he may lose contact with the previous classmates when he starts moving and then contacts other classmates after sitting down. By analyzing the collected trace data, we confirm this intuition and identify three relationships between contact changes and user movement.

Relation 1: Contact Changes vs. Start Points. We use *start point* to represent the time when a user starts to move. To show the relationship between contact changes and start points, we compute the time between contact changes and the corresponding start points and draw its distribution in Fig. 5(a). Most contact changes are close to the start points. Specifically, 50%, 80% and 90% of contact changes happen within 42, 112, and 207 seconds from the start points. From the $CCDF$ of this time in Fig. 5(b) (in log-log scale), we notice that it follows an exponential distribution in the range [6 sec, 200 sec], which means that the probability of detecting a contact change event drops exponentially as the time from the start point increases.

Relation 2: Contact Changes vs. End-Points. We use *end-point* to represent the time when a user stops moving. Fig. 6 shows the CDF of the time between contact changes and the end-points. We find 50%, 80%, and 90% of contact changes happen within 37, 83, and 124 seconds before the end-points. The number of contact changes before a given time of the end-points is shown in Fig. 6(b). Each value at time t is the number of contact changes that happen within $(t-10, t]$ seconds before the end-points. Though there is no obvious pattern, most contact changes happen very close to the end-points.

Relation 3: Communication Range vs. RSSI. To detect the end of a user contact, Bluetooth scans should be done at the boundary of the communication range. However, it is hard to determine the exact communication range since the wireless signal transmission is affected by many factors, especially in indoor environments. To address this problem, we exploit the Received Signal Strength Indication (RSSI) to estimate the communication range. Although RSSI is not the best indicator of the communication range, they are highly correlated [7]. Android provides APIs to read the Bluetooth RSSI, ranging from 0 to -100 dB, where 0 means the best signal and -100 dB the worst.

We use *lost-RSSI* to represent the last Bluetooth RSSI value a user detects about his neighbor before the end of contact with this neighbor, and then use this value to estimate the communication range. Fig. 7(a) shows that 50%, 80% and 90% of user contacts end with RSSI values -79, -69 and -62 dB. More specifically, Fig. 7(b) shows the number of user contacts which end within $(r-10, r]$ at each RSSI r. Considering 90% of the user contacts end when the RSSI is smaller than -60 dB, we pick -60 dB as the default *lost-*

RSSI threshold (α), which is also used to determine the communication range.

IV. MAUC Design

In this section, we present our Mobility-Assisted Accurate User Contact detection algorithm (MAUC), which can detect user contacts accurately with less power.

A. MAUC Overview

MAUC performs Bluetooth scans properly within the moving periods to save energy. Specifically, MAUC solves two problems: how to detect the moving periods that may cause contact changes, and how to detect user contacts within the moving periods. For each problem, we design two solutions depending on whether a user is moving or static. A moving user can detect the moving periods using accelerometer, but the static user may not be aware of such moving periods. Therefore, moving users need to notify static users about the moving periods (see Section IV-B). Based on the detected moving periods, moving users and static users apply different user contact detection algorithms, which will be presented in Section IV-C and IV-D, respectively.

B. Detecting the Moving Periods

In MAUC, a moving period is described as a set of tuples (*id*, t_{start}, t_{end}), where id is the user who causes the movement, and t_{start}, t_{end} are the corresponding start and end time of the moving period.

We use accelerometer on smartphones to detect user movement continuously and classify it into two status: *moving* or *static*, based on a decision tree technique [16], which builds decision trees from a set of training cases. Each case can be described as $(\overrightarrow{\mathbf{x}}; Y)$, where $\overrightarrow{\mathbf{x}}$ is a vector of attributes and Y is the decision. In this paper, $\overrightarrow{\mathbf{x}}$ includes the mean, standard deviation, and peak number of the three axes of accelerometer, and Y is the movement status. Note that the training process of the classifier is run off-line.

In real life, temporary vibration and shift of the smartphone may generate false moving periods, which should be filtered out. We remove all moving periods shorter than a threshold, which is set to 6 seconds based on our trace study. After filtering, the remaining moving periods are recorded as detected moving periods. For example, in Fig. 8, user B detects two moving periods, which are shown by the two rectangles above the time line.

1) Interaction between Static and Moving Users: In order to let static users know the moving periods that may cause contact changes, a moving user will initiate a Bluetooth connection with his neighbor and then disconnect, so as to notify the neighbor and save energy for data transmission [4]. When receiving such a notification, the moving neighbor just ignores it, whereas the static neighbor uses it to estimate the moving user's moving period. To save energy for both moving and static users, a moving user only

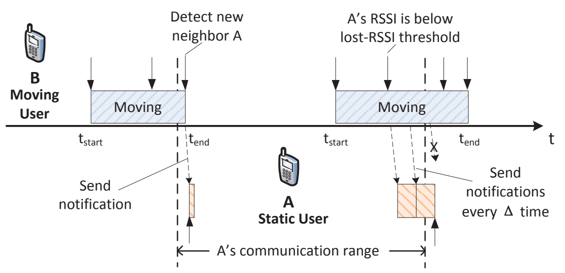


Figure 8. An illustration of MAUC. The vertical dashed line is the communication range of the static phone.

sends notification when the moving period is likely to cause contact changes in the following two cases, as shown in Fig. 8.

Case 1: A moving user B sends a notification to a new neighbor A immediately after it detects the new neighbor.

Case 2: When a moving user B moves away from neighbor A and detects A's signal is weaker than the lost-RSSI threshold (α), B sends notifications to A every Δ time, until either one of the following conditions is satisfied.

- After one Bluetooth scan, B finds A's signal is above α. Then B doesn't need to send notifications as it has moved back towards A.
- B can not detect A, as he has moved out of the communication range of A.
- B stops moving.

C. Contact Detection by the Moving User

A moving user needs to trigger Bluetooth scans properly within the moving periods to reduce the number of Bluetooth scans and improve the detection accuracy. Next, we introduce an adaptive duty cycle algorithm to coordinate the Bluetooth scans.

1) Adaptive Duty Cycle: According to Section III-C, the probability of detecting a contact change event drops exponentially from the start point of a moving period; thus, it is better to increase the scan interval quickly after a failed detection. On the other hand, the appearance of a user contact is related to the contact history [20, 18], so it is helpful to adjust the Bluetooth duty cycle considering the previous scan result. Combining them together, MAUC uses exponential-increase and multiplicative-decrease algorithm to adjust the Bluetooth scan duty cycle, as described in line 15-21 in Alg. 1. Suppose the i^{th} detection fails to detect a user contact, the scan interval will increase according to Equ. 1.

$$interval_{i+1} = interval_i \times e^{backoff} \qquad (1)$$

where backoff records the back off stage, and increases by 1 after a failed detection. After a successful contact detection, the backoff value is reset to 1 and the next scan interval will be decreased according to Equ. 2, where k is set to 2.

$$interval_{i+1} = interval_i / k \tag{2}$$

Normally, the next Bluetooth scan should start at $T_{scan} = t_i + interval_{i+1}$. However, if T_{scan} falls into a static period, there will be no Bluetooth scan and the next scan is delayed to the t_{start} of the next moving period. Another exception is the end-point detection, which may happen before T_{scan}.

2) End-Point Detection: The exponential-increase and multiplicative-decrease algorithm reduces the number of Bluetooth scans, but it also reduces the detection accuracy dramatically [18]. In Section III-C, we found that most contact changes occur near the end-points, thus we add one more detection at the end-points to improve the detection accuracy, even if it is before T_{scan}. After the detection, the scan $interval$ and T_{scan} will be adjusted based on the detection result.

According to the findings in Section III-C, the exponential decrease scheme should be used at the start of every moving period, but in MAUC we do not. The reasons are as follows. If the time interval between two moving periods is long and t_{start} of the later moving period is after T_{scan}, then there will be a detection at t_{start}. Otherwise, if the time interval is short, it would be redundant to detect again at the start point of the later moving period after the end-point detection in the previous moving period.

The detailed algorithm is shown in Alg. 1. Each moving user keeps a neighbor-set and a RSSI-set, containing the RSSI value of each neighbor. Using these two sets, a moving user can determine whether a neighbor is newly detected and whether the RSSI value of a neighbor is below the lost-RSSI threshold. Then he can notify the proper neighbors at proper time.

D. Contact Detection by the Static Users

A static user needs to detect contact changes when receiving notifications from moving neighbors. To receive notifications in time, the static user should keep the Bluetooth server on all the time. Each static user keeps a neighbor-set to determine whether a notification is from an existing neighbor or not, and a timeout-set to record the timeout value of each neighbor.

If a static user A receives a notification from neighbor B not in the neighbor-set at time t, A will record the moving period as (B, t, t). In this case A knows that some users have moved into its communication range, and then triggers a Bluetooth scan immediately to find the appearance of the new neighbors. The new neighbor is added to the neighbor-set and its timeout value is set to infinity. This process is shown in lines 6-9 in Alg. 2.

On the other hand, if a static user A receives a notification from an existing neighbor B at t, A will record the moving period as (B, t, $t + \Delta$). A knows B is near its communication range. Since this does not mean B will move out of the communication range immediately, A just updates the timeout value of B as $T_o(B) = t + \Delta$. If A receives B's notification again before the timer expires, A extends $T_o(B)$ by Δ. When the timer expires, A will start a Bluetooth scan to find out the real situation in the vicinity (line 14 in Alg. 2). Using this scheme, we can detect the loss of a moving neighbor within Δ time in the worst case.

Algorithm 1: The Contact Detection Algorithm by Moving Users

```
Input: Lost-RSSI Threshold $\alpha$
1  Initialization:
2  Neighbor-set $N \leftarrow \emptyset$
3  RSSI-set $RSSI \leftarrow \emptyset$
4  $T_{scan} \leftarrow 0, duration \leftarrow \Delta, backoff \leftarrow 1$
5  stopBluetoothServer()
6  foreach moving period $(< self >, t_{start}, t_{end})$ do
7      $EPdetect \leftarrow false$
8      $t \leftarrow t_{start}$
9      while $t < t_{end}$ or $EPdetect == false$ do
10         if $t \geq T_{scan}$ or $t \geq t_{end}$ then
11             $scanAt(t)$
12             update $N$ and $RSSI$
13             if $t \geq T_{end}$ then // End point detect
14                 $EPdetect \leftarrow true$
15             if find at least one event then
16                 $duration \leftarrow duration/k$
17                 $backoff \leftarrow 1$
18             else
19                 $duration \leftarrow duration \times e^{backoff}$
20                 $backoff \leftarrow backoff + 1$
21             $T_{scan} \leftarrow t + duration$
22             foreach new neighbor $n$ do
23                 $sendNotificationTo(n)$
24         foreach $n \in N$ do
25             if $RSSI(n) < \alpha$ then
26                 $sendNotificationTo(n)$
27                 $duration \leftarrow \Delta$
28         $t \leftarrow t + duration$
```

Algorithm 2: The Contact Detection Algorithm by Static Users

```
1  Initialization:
2  Neighbor-set $N \leftarrow$ Previous results
3  Neighbor timeout-set $T_o \leftarrow \emptyset$
4  startBluetoothServer()
5  foreach moving period $(n, t_{start}, t_{end})$ do
6      if $n \notin N$ then // Notification from new neighbor
7          $scanAt(t_{start})$
8          $N \leftarrow N \cup n$
9          $T_o(n) \leftarrow Infinity$
10     else
11         $T_o(n) \leftarrow t_{start} + \Delta$
12     foreach  $n \in N$ do
13         if $T_o(n) < now$ then // Timer expires
14             $scanAt(T_o(n))$
```

V. Energy-Aware MAUC (E-MAUC)

In the previous discussions, we assume that the Bluetooth interface keeps working in the discovery mode. Since the discovery mode is energy consuming, it is better to reduce the discovery time. In fact, the Bluetooth interface should only work in the discovery mode when other nodes are trying to detect it. In this section, we present techniques to realize this idea to save energy.

A. Adaptive Discovery

In order for smartphones to spend less time in the discovery mode, they should be aware of the discovery schedule of others and only scan at that time. This can be realized by introducing a global discovery schedule and synchronizing the smartphones using the "`Automatic`" option in the time setting menu, which sets the phone's time to that of the wireless service provider.

The most challenging problem is to let smartphones enter the discovery mode automatically according to the discovery schedule without interrupting users, which is not feasible in the original Android system. When the Bluetooth interface needs to enter *discovery* mode, a request with type `BluetoothAdapter.ACTION_REQUEST_DISCOVER-ABLE` and discoverable time will be sent to `RequestPermissionActivity.java`. The timeout value will be checked and reset to 120 seconds if it is negative. A dialog will be popped up to ask for the user's permission. If the user refuses or ignores the request, the Bluetooth interface will not be able to enter discovery according to the schedule. To solve this problem, we modify the Android OS to deal with negative timeout values. If the timeout value of a Bluetooth discovery request is $-d(d > 0)$, the request dialog will be bypassed and the Bluetooth interface is directly set to discovery for d time. When the timer expires, the discovery mode is turned off.

B. E-MAUC

Let T denote the length of the Bluetooth discovery interval. In E-MAUC, all phones will enter discovery every T time according to a global clock, and stay for T_d. If the smartphone needs to start a Bluetooth scan, it only scans at these time periods. The selection of T should not affect the detection accuracy of MAUC, so we set it to the minimum scan interval of MAUC. T_d should be longer than the maximum scan duration, which is affected by the number of neighbors, signal strength, and interference between neighbors, etc. In this paper, we set T_d to the largest scan duration based on experiments. In our trace, most scan durations are around 10 seconds and all of them are shorter than 12 seconds, so we set T_d to 12 seconds.

C. Energy Improvement of E-MAUC

The power consumed for contact detection in MAUC includes three parts: Bluetooth discovery (P_d), accelerometer (P_a), and Bluetooth scan (P_s). The total power consumption is $P = P_d + [P_a] + P_s$, where P_a is optional, depending on whether the accelerometer is used or not. Also, we have $P_s = \frac{K \times E_s}{12hour \times 3600sec/hour}$ where E_s is the energy consumed in one scan, and K is the number of scans in a day. For our phone, we have $E_s = 307.59mW \times 12s$, and hence $P_s = 0.085 \cdot K(mW)$.

In E-MAUC, the Bluetooth interface only enters discovery when necessary. Thus, the power consumption of Bluetooth discovery can be further reduced to $P_b \times (1 - \frac{T_d}{T}) + P_d \times \frac{T_d}{T}$, where P_b is the power consumption when Bluetooth is on. Therefore the power saving rate Q of E-MAUC over MAUC can be calculated using Eq. 3.

$$Q = \frac{P(\text{MAUC}) - P(\text{E-MAUC})}{P(\text{MAUC})} = \frac{(P_d - P_b) \times (1 - \frac{T_d}{T})}{P_d + P_a + P_s} \quad (3)$$

VI. Performance Evaluations

In this section, we present the evaluation results based on the collected traces.

A. Algorithms for Comparison

We compare the performance of MAUC and E-MAUC against several other contact detection algorithms.

SociableSense [18]: SociableSense adaptively adjusts the Bluetooth scan duty cycle based on whether the previous detection finds any interesting events (i.e., contact changes). After a successful detection, SociableSense adjusts the detection rate p by $p = p + \alpha(1-p)$. Otherwise, p is decreased according to $p = p - \alpha p$. Parameter α is set to 0.5, and p is restricted between 0.1 and 0.9.

STAR [20]: STAR uses the number of contacts in the previous m minutes to predict the number in the next m minutes. The duration between the two detections is adjusted according to the following Equation.

$$T = \tau(\frac{c(1-k)}{\lambda k \tau} + 1)^{\frac{1}{1-k}} \quad (4)$$

The contact arrival rate λ is calculated based on the number of user contacts in the previous m minutes. Other parameters are constant [20]. In our experiment, we set $\lambda = 1$ at the beginning of each day.

Aggressive Mobility Detection (AMD): AMD uses the most aggressive detection scheme considering user movement. During moving periods, AMD detects user contacts every 30 seconds and also applies the end-point detection scheme. During static periods, AMD detects user contacts when receiving a notification or when the timer expires. AMD can be seen as the upper bound of MAUC.

The comparisons are based on detection accuracy and energy consumption. The detection accuracy is measured by detection rate and Δ-detection rate. The energy consumption is measured by the number of Bluetooth scans and the average amount of power consumed by the whole phone. As MAUC and E-MAUC have the same detection accuracy, we only use MAUC when comparing detection accuracy.

Table II
MOBILITY CLASSIFIER CONFUSION MATRIX

	Static	Moving
Static	93%	7%
Moving	1.4%	98.6%

Table III
DETECTION ACCURACY WITH/WITHOUT END-POINT DETECTION

	detection rate	Δ-detection rate
MAUC with EP	0.78	0.57
MAUC w/o EP	0.47	0.24

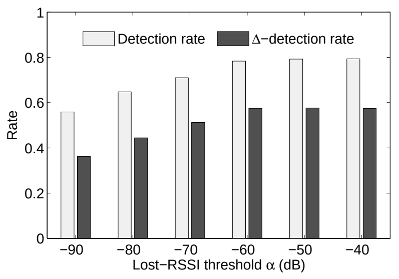


Figure 9. Impact of Lost-RSSI threshold to the detection accuracy of MAUC

B. The Effectiveness of MAUC

There are many design choices in MAUC, and we evaluate the effectiveness of the techniques used in MAUC.

1) The effectiveness of using accelerometer for movement detection: We first evaluate the effectiveness of movement detection based on accelerometer in MAUC. We asked five users to annotate their movement status every 10 minutes over two days. Then, we constructed test cases with the sampled accelerometer data as attributes and user notations as decisions. These cases are divided into two parts, one for training and another for testing. The accuracy of the movement detection algorithm used in MAUC is shown in Tab. II.

As can be seen, almost all moving periods are successfully detected. Although the false positive is as high as 7%, it just adds more Bluetooth scans, which will not affect the detection accuracy.

2) Impact of End-Point Detection: We want to evaluate whether the end-point detection helps improve the detection accuracy of MAUC. For comparison, we create another protocol, which keeps all functions in MAUC, except the end-point detection. As shown in Tab. III, with end-point detection, the contact detection rate can be increased from 47% to 78%, and the Δ-detection rate can be improved from 23% to 57%. Thus, end-point detection can significantly improve the detection accuracy of MAUC.

3) Impact of the Lost-RSSI Threshold (α): The Lost-RSSI Threshold (α) affects the performance of MAUC as it determines when to detect the potential lost of moving neighbors. Fig. 9 shows the impact of α on the detection accuracy in MAUC. Both the detection rate and Δ-detection rate increase when α increases. When α is larger than -60 dB, increasing RSSI does not bring too much improvement. The lower detection rate when α is small indicates most contact changes are not detected without correctly detecting the neighbors' leaving. It also means that observing user contact from a single user's view is not enough, since lots of user contact changes are caused by neighbors' movements.

C. Performance Comparisons

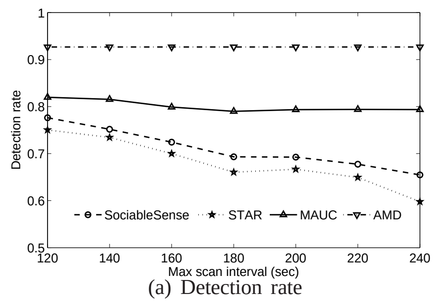
In this subsection, we compare the detection accuracy and energy consumption of various contact detection algorithms. As the performance of existing solutions are mainly affected by the scan interval, we keep the minimum scan interval to 30 seconds and increase the maximum scan interval from 120 to 240 seconds. For AMD, since it scans at a constant rate, this change will not affect its performance.

1) Detection Accuracy: As shown in Fig. 10(a) and 10(b), MAUC achieves higher detection rate and Δ-detection rate than SociableSense and STAR. When the max scan interval increases, the detection rate of SociableSense and STAR drops significantly. But MAUC can still maintain high accuracy due to end-point detection. When the maximum scan interval is 240 seconds, MAUC can improve the detection rate by 21% and 32% when compared to SociableSense and STAR, respectively.

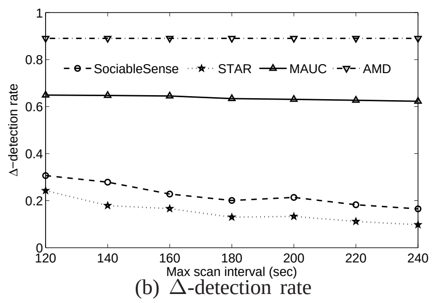
For SociableSense and STAR, we notice that the difference between their detection rate and Δ-detection rate is very large. It means that although these two algorithms detect many contacts, they can not detect the contact changes accurately, thus they can not detect contact duration accurately. In contrast, this difference of MAUC is much smaller, which shows that MAUC outperforms existing algorithms when detecting the contact changes.

The detection rate of SociableSense and STAR is very close to each other. Since user contacts are relatively rare compared to the long testing time, both algorithms tend to increase the scan interval to the maximum value. One interesting thing is that STAR performs worse than SociableSense most of time, although it uses more complex scheme to predict future user contacts.

2) Energy Efficiency: Besides achieving high detection rate, MAUC is also energy efficient. This is because MAUC does not initiate Bluetooth scans outside of the user moving periods. From Fig. 11(a), we can see that MAUC uses much less number of Bluetooth scans. When the maximum scan interval is 120 seconds, MAUC can reduce the number of Bluetooth scans by 58.1% and 57.3% when compared to SociableSense and STAR. We also compare the energy consumption of these four algorithms in Fig. 11(b). The trend is similar as Fig. 11(a), but with smaller difference between MAUC and other algorithms, because the energy saved by reducing Bluetooth scans is smaller when compared to that consumed by the Bluetooth discovery mode. Thus, we use E-MAUC to further reduce the energy consumption of Bluetooth discovery.

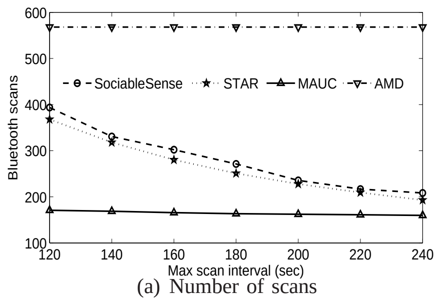


(a) Detection rate

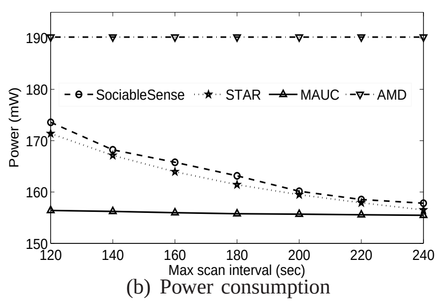


(b) Δ-detection rate

Figure 10. Detection accuracy and power comparison of MAUC and other algorithms



(a) Number of scans



(b) Power consumption

Figure 11. Detection accuracy and power comparison of MAUC and other algorithms

D. *Energy Improvement of E-MAUC*

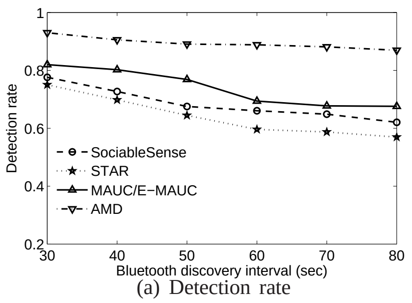
E-MAUC lets the Bluetooth interface enter the discovery mode only during the scan interval and turn off the discovery mode during other times. Thus, it can save more energy while maintaining the same detection accuracy as MAUC. Fig. 12 shows the detection rate and the power consumption of various algorithms when the discovery interval T increases from 30 to 80 seconds. For each T, we set the minimum and maximum scan interval to T and $4 \times T$. As shown in Fig. 12(a), increasing the Bluetooth discovery interval reduces the detection rate of all algorithms, but MAUC/E-MAUC still outperforms SociableSense and STAR. Fig. 12(b) shows the energy consumption of smartphones using various algorithms. As can be seen, E-MAUC can significantly reduce the energy consumption since it turns off the discovery mode most of time. When T is 30 second, E-MAUC saves 45%, 49.2% and 48.7% more energy when compared to MAUC, SociableSense and STAR, respectively.

VII. Related Work

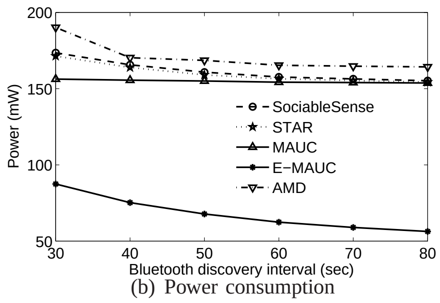
There have been many solutions on detecting user contacts in mobile social networks or mobile opportunistic networks. In [11], nodes connected to the same access point via WiFi are classified as contacting each other. However, two nodes may be out of communication range if they are at different side of the access point. In [19], class schedules are used to infer student contacts, but the contacts after class or between classes cannot be modeled accurately. A more realistic approach to detect user contact is based on Bluetooth due to its short communication range, and it has already been used in several projects [6, 2]. Among them, the Infocom trace [6] is collected with people attending a conference, and the MIT Reality trace [2] is based on students on campus. However, none of these works consider energy issues.

To detect user contacts with Bluetooth, power consuming Bluetooth scans must be used. Although reducing the number of scans can save energy, it also reduces the detection accuracy [15, 17, 9, 13]. To address this problem, researchers in [20] models user contact durations with Pareto distribution, and propose a scheme called STAR to reduce the number of Bluetooth scans based on this distribution. Similar solutions have been proposed in [21], which uses Markov Model to describe user state changes and predict the next contact. Rachuri *et al.* [18] introduce a heuristic algorithm to adjust the duty cycle using linear reward inaction algorithm based on previous detection results. However, the predictions in these algorithms may be wrong, and then result in low detection accuracy, as shown in our evaluations.

Different from these existing works, we address this problem based on the observation that user contact changes (i.e., starts and ends of user contacts) are mainly caused by user movement. Since most smartphones have accelerometers, we can use them to detect user movement with much less energy and then start Bluetooth scans to detect user contacts. Using accelerometers to detect user mobility has been around for some time [8, 1]. For example, accelerometers are used to detect user moving speed and moving distance in [1] and user activity in [12]. However, none



(a) Detection rate



(b) Power consumption

Figure 12. Performance comparisons with various discovery interval

of them has been applied to saving energy in user contact detection. Although it is generally not very accurate for using accelerometers to detect fine-grained user activity, this is fine for our application since we only need to determine whether the user is moving or not. Moreover, to improve the detection accuracy, we also identified the relationships between contact changes and user movement, and proposed techniques to improve the detection accuracy such as using end-point detection, etc.

VIII. Conclusions

In this paper, we proposed a Mobility-Assisted User Contact detect algorithm algorithm (MAUC) to detect user contacts with high accuracy and low energy consumption. MAUC uses accelerometers to detect user movements and only start the power consuming Bluetooth scans when users or their neighbors are moving. To improve the detection accuracy, we identified the relationships between contact changes and user movement based on our collected real traces, and proposed techniques to improve the detection accuracy such as using end-point detection, etc. For a Bluetooth device to be detected through Bluetooth scan, it must stay in the Bluetooth discovery (discoverable) mode, which also consumes lots of power. To address this problem, we propose energy-aware MAUC (E-MAUC) to reduce the amount of time that a Bluetooth device has to stay in the discovery mode. Via trace driven simulations, we show that MAUC can reduce the number of Bluetooth scans by half while maintaining similar detection rate with existing algorithms, and E-MAUC can further reduce the energy consumption by 45% compared to MAUC.

References

[1] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury. Did you see bob?: human localization using mobile phones. In *ACM Mobicom*, 2010.

[2] N. Eagle and A. (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4), Mar. 2006.

[3] E. Fleury, J.-L. Guillaume, C. Robardet, and A. Scherrer. Analysis of dynamic sensor networks: Power law then what? In *2nd International Conference on Communication Systems Software and Middleware*, 2007.

[4] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. In *IEEE Infocom*, 2011.

[5] W. Gao, G. Cao, T. La Porta, and J. Han. On exploiting transient social contact patterns for data forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 12(1):151–165, 2013.

[6] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *ACM SIGCOMM workshop on Delay-tolerant networking*, 2005.

[7] R. Jurdak, P. Corke, D. Dharman, and G. Salagnac. Adaptive gps duty cycling and radio ranging for energy-efficient localization. In *ACM Sensys*, 2010.

[8] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava. Sensloc: sensing everyday places and paths using less energy. In *ACM Sensys*, 2010.

[9] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *ACM Mobisys*, 2010.

[10] Z. Lu, Y. Wen, and G. Cao. Community detection in weighted networks: Algorithms and applications. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2013.

[11] M. McNett and G. M. Voelker. Access and mobility of wireless pda users. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(2), Apr. 2005.

[12] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *ACM Sensys*, 2008.

[13] S. Nath. Ace: exploiting correlation for energy-efficient and continuous context sensing. In *ACM MobiSys*, 2012.

[14] N. P. Nguyen, T. N. Dinh, S. Tokala, and M. T. Thai. Overlapping communities in dynamic networks: their detection and mobile applications. In *ACM Mobicom*, 2011.

[15] B. Priyantha, D. Lymberopoulos, and J. Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *Pervasive Computing, IEEE*, Feb. 2011.

[16] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[17] M.-R. Ra, J. Paek, R. G. Abhishek B. Sharma, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *ACM Mobisys*, 2010.

[18] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *ACM MobiCom*, 2011.

[19] V. Srinivasan, M. Motani, and W. T. Ooi. Analysis and implications of student contact patterns derived from campus schedules. In *ACM Mobicom*, 2006.

[20] W. Wang, V. Srinivasan, and M. Motani. Adaptive contact probing mechanisms for delay tolerant applications. In *ACM Mobicom*, 2007.

[21] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Markov-optimal sensing policy for user state estimation in mobile devices. In *ACM/IEEE IPSN*, 2010.

[22] Q. Yuan, I. Cardei, and J. Wu. Predict and relay: an efficient routing in disruption-tolerant networks. In *ACM Mobihoc*, 2009.

[23] B. Zhao, B. C. Tak, and G. Cao. Reducing the delay and power consumption of web browsing on smartphones in 3g networks. In *IEEE ICDCS*, 2011.

[24] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli. Energy-aware web browsing in 3g based smartphones. In *IEEE ICDCS*, 2013.

[25] X. Zhuo, Q. Li, W. Gao, G. Cao, and Y. Dai. Contact duration aware data replication in delay tolerant networks. In *IEEE ICNP*, 2011.