Ye HB, Gu T, Tao XP *et al.* Infrastructure-free floor localization through crowdsourcing. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(6): 1249–1273 Nov. 2015. DOI 10.1007/s11390-015-1597-z

Infrastructure-Free Floor Localization Through Crowdsourcing

Hai-Bo Ye [1,2] (叶海波), Tao Gu [3] (顾 涛), *Senior Member, IEEE, Member, ACM*
Xian-Ping Tao [1,2] (陶先平), *Member, CCF, IEEE*, and
Jian Lv [1,2] (吕 建), *Fellow, CCF, Member, ACM*

[1] *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[2] *Department of Computer Science, Nanjing University, Nanjing 210023, China*
[3] *School of Computer Science and Information Technology, RMIT University, Melbourne, Victoria 3001, Australia*

E-mail: yhb@smail.nju.edu.cn; tao.gu@rmit.edu.au; {txp, lj}@nju.edu.cn

Received July 16, 2014; revised May 13, 2015.

Abstract Mobile phone localization plays a key role in the fast-growing location-based applications domain. Most of the existing localization schemes rely on infrastructure support such as GSM, Wi-Fi or GPS. In this paper, we present FTrack, a novel floor localization system to identify the floor level in a multi-floor building on which a mobile user is located. FTrack uses the mobile phone's sensors only without any infrastructure support. It does not require any prior knowledge of the building such as floor height or floor levels. Through crowdsourcing, FTrack builds a mapping table which contains the magnetic field signature of users taking the elevator/escalator or walking on the stairs between any two floors. The table can then be used for mobile users to pinpoint their current floor levels. We conduct both simulation and field studies to demonstrate the efficiency, scalability and robustness of FTrack. Our field trial shows that FTrack achieves an accuracy of over 96% in three different buildings.

Keywords mobile phone localization, floor localization, crowdsourcing, mobile phone sensing

1 Introduction

With the increasing pervasiveness of mobile phones, we have witnessed the rapid growth of location-based applications (LBAs) in recent years. Such applications can be micro-blogging, location-based advertising[1-2], local traffic[3-4], etc. In a multi-floor building environment, knowing the floor level of a mobile user is particularly useful for a variety of LBAs. For example, in location-based advertising, advertisements can be delivered to mobile users based on their floor levels in a shopping mall. In emergency situations, locating the floor level of a user fast and accurately can be very critical for emergency services such as medical assistance or life saving. The recently launched Google Maps 8.0 for Android offers a new feature named Go indoors. Go indoors provides users with the detailed floor plan of a building (shopping center, airport, etc.) according to the floor level they are currently on. This problem is known as the floor localization problem, determining the floor level in a building on which a mobile user is located.

Localization has attracted extensive research efforts in recent years. A variety of indoor localization methods[5-7] have been proposed. Techniques based on radio signals from Wi-Fi access points or GSM base stations have shown promising results. RADAR[6] operates on Wi-Fi fingerprints, and it is capable of achieving an accuracy of up to 5 m in indoor environments. However, even such an idealized scheme may not be adequate to identify the floor level accurately because it can easily lead to confusion between two neighbor floors. PlaceLab[5] uses Wi-Fi and GSM signals. The idea is to war-drive an area to create a radio map of existing Wi-Fi/GSM access points, and a mobile phone localizes itself by comparing overheard Wi-Fi/GSM ac-

Regular Paper

This work was supported by the National High Technology Research and Development 863 Program of China under Grant No. 2013AA01A213 and the National Natural Science Foundation of China under Grant Nos. 91318301, 61373011 and 61321491.

A preliminary version of the paper was published in the Proceedings of PerCom 2012.

cess points against the radio map. However, the limitations of these methods have been well studied.

First, Wi-Fi/GSM-based localization relies on pre-installed infrastructure under dense deployment. Such infrastructures may not be available in many buildings. Especially, in developing countries, many buildings have no Wi-Fi or sparse Wi-Fi coverage which is not dense enough for localization. Even in developed countries, a study① shows Wi-Fi may not be fully available in many buildings. An infrastructure-free solution would be highly desirable for floor localization. Second, existing methods typically require calibration which is expensive and not scalable. RADAR needs to carefully calibrate Wi-Fi signal strengths at many locations, and the calibration process is time-consuming and may not scale over large areas. War-driving in PlaceLab is also intolerably time-consuming in order to cover a large portion of space. The recurring cost in war-driving can be excessive and undesirable. Some recent approaches such as LiFS[8] use crowdsourcing to reduce the war-driving cost to some extent, but it involves a complicated training process. An ideal solution should be cheap and scalable.

A variety of sensors embedded in smartphones have recently motivated the research on sensor-assisted localization methods[9-11]. The basic idea is to leverage the mobile phone's accelerometer and electronic compass to measure the walking speed and orientation of the mobile user. The user's location can be easily computed by double-integrating the acceleration readings over time. However, electronic compasses and accelerometers are highly noisy[12], causing the user location to diverge from the actual location. Constandache *et al.*[9-10] proposed a better approach by identifying acceleration signatures in human walking patterns (i.e., the nature up and down bounce), and then multiplying step count with the user's step size to obtain the location. However, Escort[9] leverages fixed beacons for constant calibration, and CompAcc[10] makes use of possible walking paths extracted from Google Maps②. Crowdsourcing has also been used to reduce the war-driving effort[13-14]. These studies rely on detecting user activities using sensors such as accelerometer. However, to ensure reliable detection, they typically require user-specific training which is costly, and high sampling frequency which may drain the battery power quickly. In addition, the detection may be often interrupted by users making or receiving phone calls.

A new fingerprinting approach based on magnetometer on smartphones has been proposed[15-17] recently. The abnormalities of the magnetic field can be used as fingerprints for indoor localization. While these approaches share a similar idea to Wi-Fi fingerprinting, they need even more war-driving cost.

Paper [18] shows the properties of mobile-embedded barometers across a number of buildings. The authors concluded that it is difficult to use the barometer to determine the actual floor that a user is on. In another typical solution proposed by Wang *et al.* in [19], they tracked the user using barometer readings. First, they assumed the user's initial floor level is f_0. When the user changes floor levels, the barometer reading change Δp can be detected, and the user's new floor level is computed as $f_0 + \Delta p/(0.12 \times h_0)$. In reality, h_0 varies from buildings to buildings, and each floor may have different heights, so the floor height of each floor of every building is needed, limiting the scalability of this approach. More importantly, the initial floor f_0 is difficult to know, because users may not always enter into a building from the ground floor and they may start using the localization service at any floor level. Furthermore, a miss or wrong detection of Δp will cause serious errors in the latter localization.

In this paper, we present FTrack, an infrastructure-free floor localization scheme. When doing localization, FTrack makes use of the mobile phone's magnetometer and barometer only, and does not require any infrastructure support (i.e., Wi-Fi/GSM). Using magnetometer data, our solution works based on the following observation. When users walk on the stairs from floor i to floor j in the same stairwell, the magnetometer data traces scanned from their smartphones show the same unique signature due to a variety of physical environments in different stairwells. Such signature can be exploited as fingerprints to locate the user by pattern matching. This also holds when people take the same elevator/escalator between floors. Provided a mapping table which maps the magnetic field signature to floors, we can locate a user's floor level by looking up the table. In order to get the mapping table, war-driving is a common approach. However, war-driving to get the magnetic field signature is costly, especially for tall buildings. In our approach, we propose a novel solution in FTrack which can build the mapping table automatically through crowdsourcing.

①Wifi usage across the world. http://imgur.com/gallery/mA2SyFv, Sept. 2015.

②http://maps.google.com/, July, 2014.

Our idea works as follows. For every building, when people change their floor, we can detect the floor change activity based on barometer on smartphones, but which floor they locate in is unknown. Using the start and stop time of the floor change activity, we get the magnetometer data trace signature when the user is moving between floors. After comparing it to the magnetic field signature in the mapping table, we get the floor levels where the user leaves and arrives. FTrack creates the mapping table through crowdsourcing. In a building environment, people often encounter each other on any floor or in the elevators. User trails are logged when they travel. By examining information such as user encounters, going up/down, and users joining and leaving in the trails, FTrack generates the mapping table, which can be then used for mobile users to pinpoint their current floor levels.

In summary, this paper makes the following contributions.

• We propose a novel floor localization system, named FTrack, which identifies the floor level on which a mobile user is located. FTrack makes use of the mobile phone's magnetometer and barometer only, and it does not rely on any infrastructure such as Wi-Fi/GSM.

• Through crowdsourcing, we design our algorithms to build a magnetic field signature mapping table, without any prior knowledge of the building (i.e., number of floors and floor height). In addition, we propose an algorithm to remove the inconsistency caused by different fault cases, aiming for improving the robustness of FTrack. We show theoretically that our algorithms are both feasible and scalable.

• We conduct comprehensive simulations using a realistic user mobility model to analyze FTrack. We also conduct a field study in three different buildings, and the results show that FTrack achieves an accuracy of 96%.

The rest of this paper is organized as follows. Section 2 presents an overview of FTrack, followed by the detailed design in Section 3. Section 4 presents our theoretical analysis, and Section 5 describes our evaluation. Section 6 discusses the related work, and finally, Section 7 concludes the paper.

2 System Overview

We give an overview of our system in this section, as shown in Fig.1. The system operates in two phases. In the first phase, FTrack builds the mapping table. When users travel in the building, mobile clients collect and transfer their trails to the cloud server. User trails

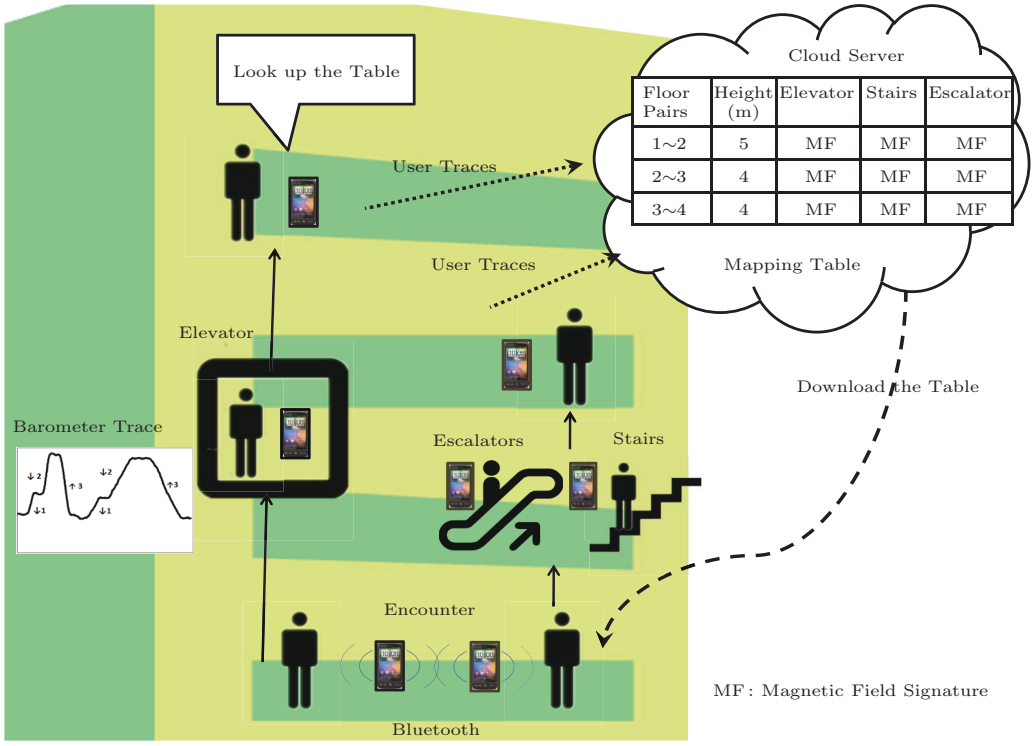


Fig.1. Overview of FTrack.

contain the information about user encounters, going up/down, the traveling distance between floors and the magnetometer data trace when the user is moving between floors. The server runs our algorithms to build the table which contains the magnetic field signature between any two floors. With enough trails collected, we can obtain the signature for any two floors. In the second phase, mobile clients can extract the magnetic field signature when the user is moving between floors, and pinpoint the current floor by comparing the magnetic field signature with the signatures in the mapping table.

In the first phase, FTrack recognizes user activities of changing floors using barometer. Based on these detected activities, we define two states of the mobile user, a moving (M) state when a user moves up/down between floors, and a static (S) state when a user stays on a floor, including the situation which the elevator stops temporarily on a floor. The states of a user may change over time. User trail is defined as a sequence of state changes. For example, if user i enters the building from the ground floor, takes the elevator to the 5th floor, visits some shops in the same floor, and then walks up to the 6th floor, the user trail will be recorded as $T_i = (S_1, M_1, S_2, M_2, S_3)$.

Trails from different people are collected and transferred to the cloud server, including the user encounter information detected by Bluetooth. We design an algorithm to first group people from the same floor where they encounter each other, and then sort all the groups by examining the sequence of state changes in the user trails. Finally, each group can be mapped into a unique floor level, and a series of tuples, i.e., $(FromFloor, ToFloor, Distance, MF\_Sig)$ is stored in the table, as shown in Fig.1 (i.e., the table in the cloud server). MF_Sig contains magnetic field signatures of all ways (stairs, elevators and escalators) that a user can go from $FromFloor$ to $ToFloor$. For a building with f floors, the number of floor pair (i, j) is $f(f-1)/2$. Actually, FTrack can locate users when having magnetic field signatures of adjacent floor pairs, for example, floor pair $(i, i+1)$. The signature of floor pair (i, j) is gotten by the merge of floor pairs $(i, i+1), ..., (j-1, j)$. The algorithm runs incrementally with new user trails until all the tuples are found. That is because an original signature of floor pair (i, j) provides a better accuracy than a merged signature of floor pair (i, j).

In the second phase, to access FTrack, mobile clients download the mapping table from the server. When the user travels in the building, the floor change activity can be detected and the magnetic field signature can be extracted using the mobile phone's barometer. After comparing it with the signatures in the mapping table based on DTW[20], FTrack can get users' floor levels.

It is worth knowing that the first phase to generate the mapping table is once for all, and the mapping table can be used forever for this building. In the second phase, localization is done locally at mobile clients. In this way, when doing floor localization, FTrack provides better user privacy protection compared with the existing approaches[10-11] which rely on the server to compute their locations.

3 System Design

We now describe the system design. The first phase of FTrack consists of the following stages — in-building state, state recognition, user encounter detection, user trails collection, grouping and merging, and mapping table generation.

3.1 In-Building State

To bootstrap FTrack, we first need to know whether a user is in a building. If he/she is in a building, we are required to know the building. We show here how to detect in which building a user is in each phase of FTrack.

In the first phase, FTrack wants to build the mapping table of a building through crowdsourcing. Here, we adopt a hybrid approach to know which building a user is in. We make use of the GPS, light and magnetometer sensors on the smartphone. The main idea is that when a user is in outdoors, the GPS gives us the continuous and accurate location of the user. After a user moves from outdoors to indoors, the last known GPS location should be just around the building. By looking up the map service, we can find a building based on this location, and we conclude where the user is. Next, we will show the detail of this approach. Before that, we admit that our approach of indoor/outdoor detection is not very energy efficient. It is important to notice that the first phase to build the mapping table is once for all and can be finished in a short time, making this energy consumption acceptable.

The light sensor can help outdoor/indoor detection. Our primary observation for light detector is that the light intensity inside buildings is typically much lower than that in the outdoor environment even in cloudy or rainy days. Fig.2 shows a clear signature based on the

light strength measurements between indoor and outdoor in different situations. Such a phenomenon can still be observed when the light sensor is rotated towards the ground.

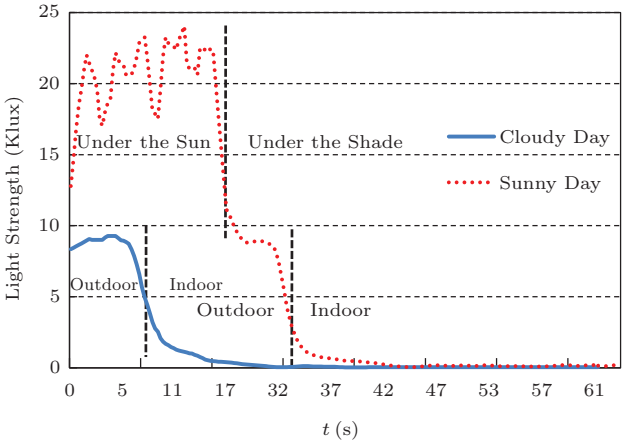


Fig.2. Light strength for indoor and outdoor.

The magnetic field readings also show different patterns in indoor/outdoor scenarios. The value of magnetic fields shows a higher variance across different places inside buildings than that in the open space. Figs.3 and 4 plot the magnetic field intensity and its variance in an example scenario respectively in which a user walks outside the building. In Fig.3, we find that the intensity of magnetic field in the indoor environment varies dramatically. Fig.4 plots the variance averaged over t seconds to filter out noises. We find that the variance is very high in the first 60 seconds when the user moves in the indoor environment. In contrary, after the user is in the outdoor environment from 61 to 140 seconds, the variance drops significantly. Therefore, by choosing a suitable threshold, we could distinguish indoor from outdoor.

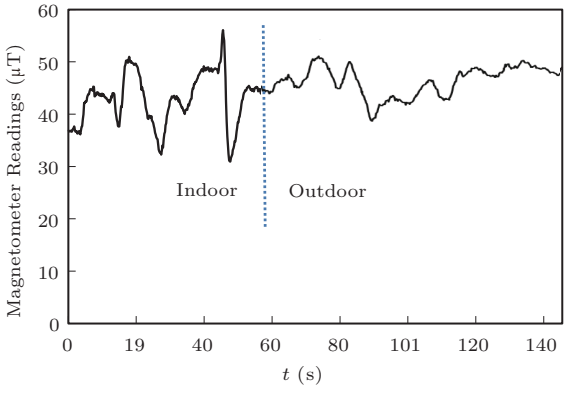


Fig.3. Magnetic field reading from indoor to outdoor.

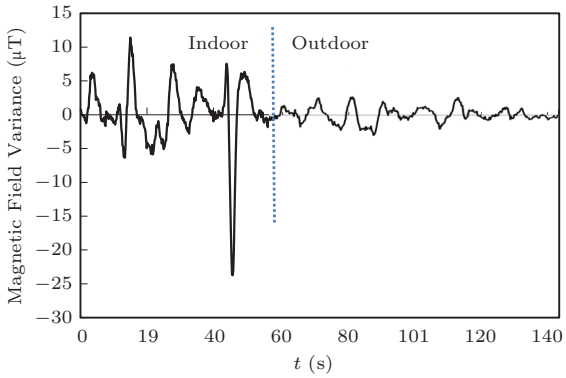


Fig.4. Magnetic field variance from indoor to outdoor.

The GPS signal is another way we use for indoor/outdoor detection. Fig.5 plots the number of visible satellites as well as the SNR (signal to noise ratio) of the GPS signals in the indoor and the outdoor environments, respectively. We can see that in the indoor environment the smartphones normally find less than two satellites, and sometimes can get slightly more GPS signals near windows. In the outdoor environment, the smartphones normally receive signals from more than six GPS satellites. Fig.5(b) plots the SNR of the received GPS signals. The SNR value in the indoor environment varies from 0 to 10, much less than the outdoor environment (i.e., 25 to 42).

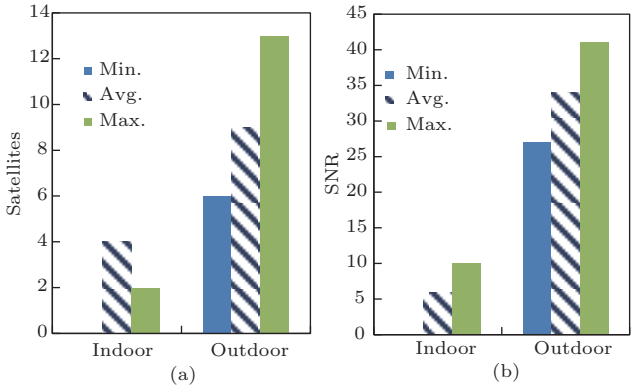


Fig.5. Number of visible satellites and SNR. (a) Number of satellites. (b) Signal to noise ratio.

In our approach, we use a hybrid way which integrates the signatures of GPS, light and magnetometer sensors of the smartphone. In detail, for a timestamp of every 10 seconds, the FTrack client will get a GPS location and run an indoor/outdoor detection based on GPS, light and magnetometer readings. The final indoor/outdoor result will be the combination of the three outputs. Once an indoor result is detected, the FTrack

client will use the latest valid GPS location to find out which building the user is in.

In the second phase, users use FTrack to locate themselves. This process happens very often and requires an energy efficient way for indoor/outdoor detection. It is based on the following observation that our way of pattern matching approach based on the magnetic field signature can directly locate the user into the right building. In this way, we can simultaneously solve the problems of finding the building and finding the floor.

3.2 User State Recognition

To detect user states (i.e., S or M) and their moving directions (i.e., up or down) at any time, we use the mobile phone's barometer.

3.2.1 Barometer on Smartphone

We now move to study the barometer sensor on smartphones. The barometer sensor has become increasingly popular on smartphones today. Most commonly used barometer sensors are BMP180/182 and LPS331AP. Table 1 gives their technical specifications. From the table, we observe that while the absolute accuracy (the accuracy of a sensor reading compared with the real barometric pressure) is about ± 20 meters (which is low), the relative accuracy (the accuracy of the change of a sensor reading compared with the change of real barometric pressure) is high. This implies that the barometer sensor has a high level of sensitivity, and it is good enough to detect the change of the barometric pressure when users go up or down in a building. Motivated by this observation, we use barometer to detect the activities when users change their floor levels.

We use a professional digital pressure gauge to measure the barometric pressure at a fixed location in an office building over a period of half an hour. Fig.6(a) plots the result. From the figure, we observe the barometric pressure measurements changing with a variation of 1.2 hPa which is equivalent to about 10 meters in altitude. This variation may result in a detection error ranging up to three floor levels. Hence, directly applying the barometric formula to calculate the floor level is not feasible. In another study, we sample the barometer readings of two smartphones of the same type at the same indoor location. Fig.6(b) shows that a constant drift of sensor readings which may result in an error ranging up to three floor levels. That confirms the barometer readings have low absolute accuracy.

3.2.2 Floor Change Activity Detection

We first present a novel technique to recognize the activities of changing floors using barometer. We represent a barometer sample P by $B = (t, Baro)$, where t is the time for sampling, and $Baro$ is the barometer reading at time t. The barometer samples arriving in time order form a barometer trace, which is represented by $BTrace = (ID, B_1, B_2, ...)$, where ID is the identity of the user. Users typically change their floor levels by taking elevators/escalators and walking up or down the stairs. The barometer sensor is inherent noisy. Fig.6(d) shows the raw barometer readings which apparently contain spike noise. In B-Loc, we first filter the noise, and then smooth the values with a reasonable window size of 1 000 ms (i.e., the value at time t is the average value from $t - 500$ ms to $t + 500$ ms), as shown in Fig.6(e). In our previous study, we observe that barometer readings on smartphones do not change much in a short period of time unless users change their floor levels. Hence, the change of barometer readings can be used to recognize the floor change activities. To do this, we extract the first derivative of the barometer readings and the resulting curve is shown in Fig.6(f). We can see from Fig.6(f) that the change of barometer readings is transformed to the crest when going up and the trough when going down. The crest and the trough are sharp when taking elevators and smooth when taking escalators and stairs. The start time (st) and the end time (et) of the activity are the time of the left and the right edge of each crest or trough respectively.

To detect these activities, we calculate the area size of each crest or trough. If it meets certain conditions, a floor change activity is detected. In detail, each area is defined as a continuous and closed region formed by the x axis and the curve. The region is located below or upon the x axis, which

Table 1. Barometer Sensor Parameters

	Absolute Accuracy	Relative Accuracy	Noise	Used in Smartphone
BMP180/182	−4.0∼+2.0 hPa ([−33 m, +17 m])	±0.12 hPa (±1 m)	0.06 hPa (0.5 m)	Galaxy Note 2/3, Xiaomi M2, Sony Ericsson Active, Nexus 3/4
LPS331AP	−3.2∼+2.6 hPa ([−27 m, +22 m])	±0.2 hPa (±1.7 m)	0.06 hPa (0.5 m)	Galaxy S3, S4

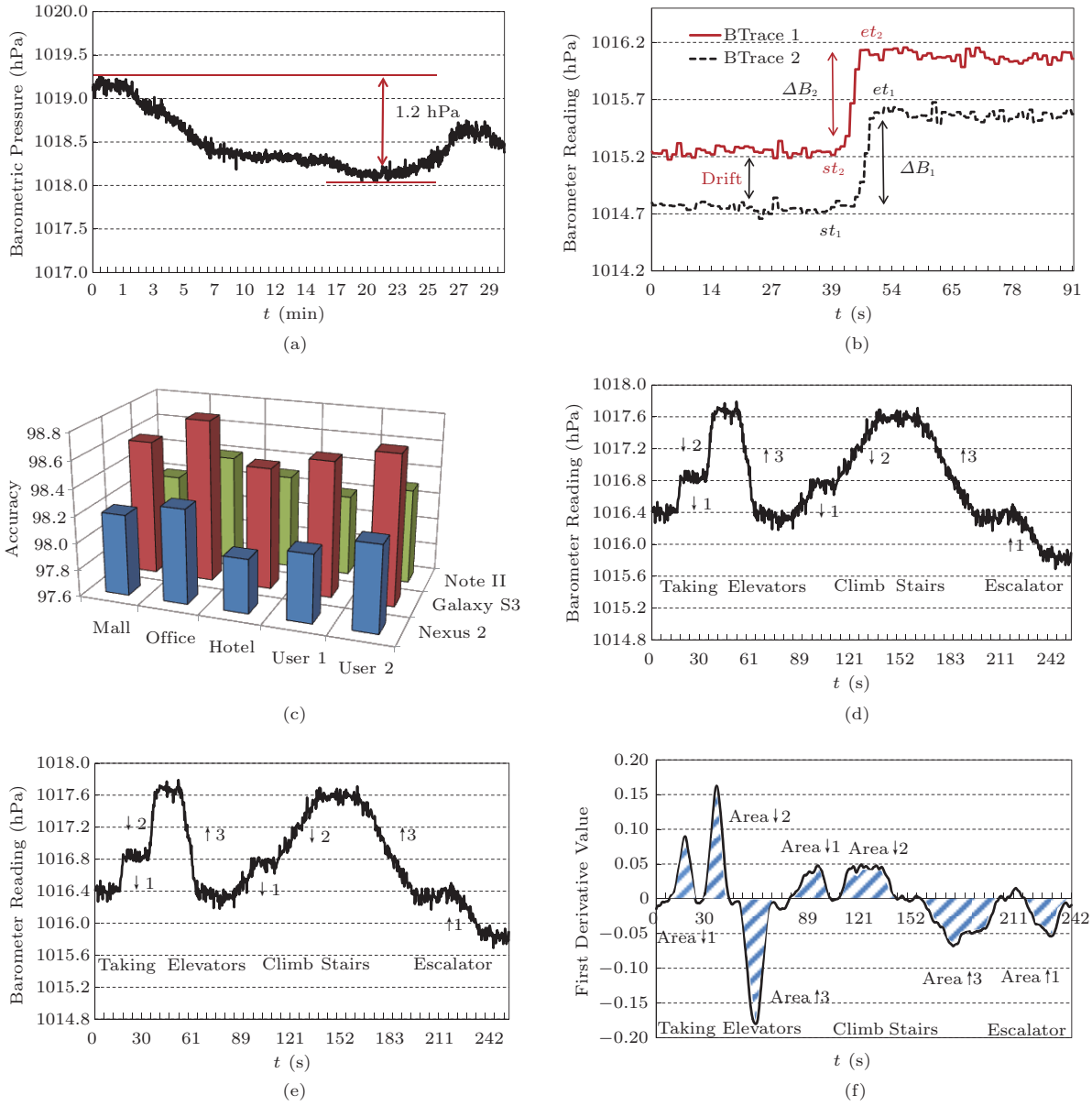


Fig. 6. Properties of smartphone's barometer and floor change activity detection by barometer readings. (a) Barometric pressure changes by time. (b) Barometer reading drift between smartphones. (c) Floor change detection accuracy. (d) Raw barometer readings. (e) Smoothed barometer readings. (f) First derivative value of the readings.

should meet the following conditions: 1) lasted time between 3 and 120 seconds, and 2) area size bigger than 1.0. Fig.6(f) shows the areas of different ways of floor changing. We define a floor change activity as $A = (STime, ETime, SBaro, EBaro)$, where $STime$ is the start time of an activity, $ETime$ is the stop time of an activity, and $SBaro$ and $EBaro$ are the barometer reading at $STime$ and $ETime$, respectively.

In B-Loc, we do not impose any constraint on the ways users carry or use their smartphones. A smartphone can be held on hand, placed into a pocket or bag, or used to make/receive a phone call, etc. This certainly offers a great advantage over the accelerator-based activity recognition[13]. We conduct experiments with two users using three different smartphones under real-life situations in three different buildings. Fig.6(c) shows the accuracy of detecting floor changes. The results show the average accuracy using barometer is about 98.3%.

It is worth knowing that barometer readings at this stage are used for real-time floor change activity detection on smartphones. After detecting a floor change

activity, we can get the magnetometer data trace when the user is moving between floors (from $A.STime$ to $A.ETime$), but the specific floors are unknown. If we know the floors, we can easily build the mapping table by extracting the magnetic field signature from the magnetometer data trace. We will show at a later stage (Subsection 3.5) how to get the floor levels of the floor change activity.

3.3 User Encounter Detection

In a multi-floor building, people often encounter each other in the elevators, or on any floor level. FTrack detects user encounters for any user who is in the S state, i.e., when users are staying on any floor level. To detect encounters, we take a simple approach by measuring the received signal strength indicator (RSSI) of the Bluetooth radio on the mobile phone. When the RSSI level grows higher than a threshold, we can conclude that the two users are in close proximity (i.e., they encounter each other). The device-to-device communication is not needed. We just need the Bluetooth open. An open Bluetooth will periodically scan the devices around, get the ID and receive the signal strength of the devices available. The approach guarantees a high true positive (TP) rate (i.e., the encounters detected are true encounters). Although there may be some missing detections due to signal variation in indoor environments, it does not affect the operation of FTrack. Another approach[9], detecting encounters by playing and listening to a specific tone at inaudible frequencies, can be used at the price of complexity and computation cost.

Now more and more smartphones support Bluetooth 4.0, and the Bluetooth Low Energy (BLE) technology can provide considerably reduced power consumption and cost while maintaining a similar communication range as before, making our solution more efficient. We test the performance of our encounter detection approach by the following experiment. Three users are walking randomly in two floors, and each user carries two mobile phones: one in the pocket for encounter detection, and the other one held in hand for the ground truth recording. For the result in Fig.7, there are totally 108 real encounters, 80 detected, and 28 missed, which indicates an accuracy of 72%. This result does not seem good, but is enough for FTrack. We do not require to detect all encounters.

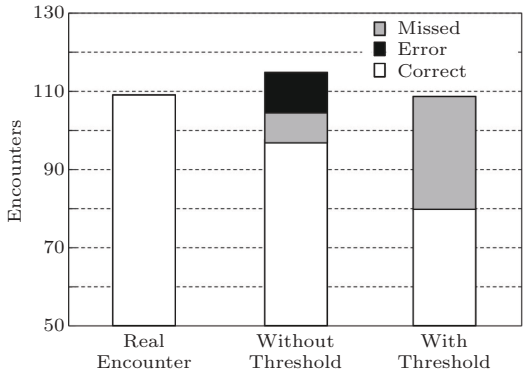


Fig.7. Encounter detection accuracy.

3.4 User Trails

FTrack records user trail as a sequence of the states of S and M, containing four types of information. They are user encounters, going up/down, the traveling distance between floors and the magnetometer data trace when the user is moving between floors. User trails typically begin and stop with the S state (since a user must be present on a floor when entering a building).

The structure of the S state is defined as $(UserID, StartTime, EndTime, Encounters)$, where $UserID$ is the ID of the user, $StartTime$ and $EndTime$ are the start and the end time of the S state, and $Encounters$ is a structure which records user encounters if any. $Encounters$ is defined as a set of $(ID, Time)$, where ID is the encountered user's ID and $Time$ is the time when the encounter occurs. When we detect a floor change activity A, we get an M state. The structure of the M state is defined as $(UserID, Distance, Direction, NTrace)$, where $UserID$ is the ID of the user, and $Distance$ is the altitude change when the user changes the floors, which can be computed by $|A.SBaro - A.EBaro|/0.12$. $Direction$ is the direction of the floor change activity which can be Up (going up when $A.SBaro > A.EBaro$) or $Down$ (going down when $A.SBaro < A.EBaro$). $NTrace$ is the magnetometer data trace collected when the user is moving between floors (from $A.STime$ to $A.ETime$). We represent a magnetometer data sample N by $N = (t, m)$, where t is the time for sampling, and m is the norm of the magnetometer readings vector in three directions at time t. The magnetometer samples arriving in time order form a magnetometer trace, which is represented by $NTrace = (ID, N_1, N_2, ...)$, where ID is the identity of the user.

User trails are collected by mobile clients and transferred to the server periodically.

3.5 Grouping and Merging

The historical information contained in user trails is used to infer the number of the floors. We illustrate this process using an example, as shown in Fig.8. Assume that we have five users in a 5-floor building, at the starting point, users 1, 3 and 5 are on the ground floor, user 2 is on the 3rd level, and user 4 is on the 2nd level. User 1 takes the elevator from the ground floor to the 5th floor level, while the elevator stops on the 3rd and the 4th levels, respectively. User 1 encounters user 3 on the ground floor, users 2 and 3 in the elevator on the 3rd level, user 2 in the elevator on the 4th level, and finally user 5 in the elevator on the 5th level. If any two users who are in the S state encounter each other, it is obvious that they are on the same floor. Based on this rule, we generate many groups, indicated by dotted lines in Fig.8. By knowing the sequence of members joining and leaving a group, we can infer whether a group is at a higher level than another. We can then sort all the groups from the lowest to the highest level. Finally, each group can be mapped to a corresponding floor.

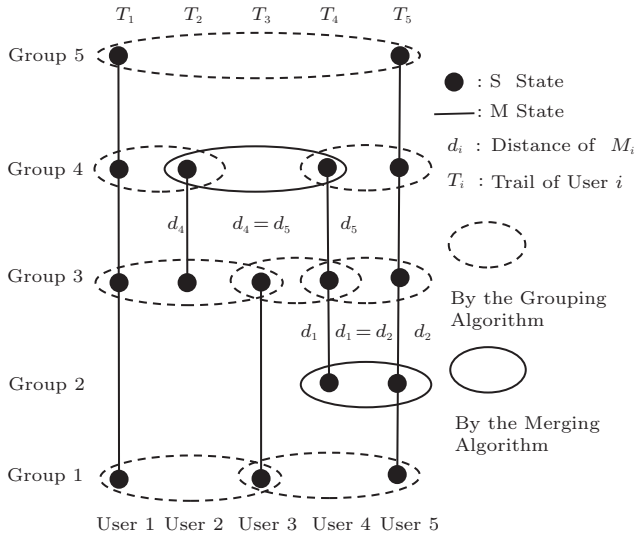


Fig.8. Grouping and merging.

Finding the total number of floors depends on the number of trails. The more the segments we collect, the faster the result we obtain. In a multi-floor building, if there are users on each level and at least one group on each level (we name the trails collected complete trails), we can ensure the completeness and correctness of the result with a high probability (we will show our analysis in Subsection 4.3). However, collecting the complete trails takes time which may be longer in a building with less crowds. To accelerate the grouping process, we propose the merging algorithm, which is capable of inferring additional knowledge based on the information contained in the trails. The algorithm is designed according to the following rules.

Rule 1. *For people on the same floor initially, after they travel towards the same direction (i.e., up or down, maybe at different time) for the same distance, they must end up on the same floor.*

To formulate the rule, we define the following propositions.

$A_1 : \exists\{S_m, M_m, S_{m+1}\} \subseteq TS_i$,
$A_2 : \exists\{S_n, M_n, S_{n+1}\} \subseteq TS_j$,
$A_3 : M_m.distance = M_n.distance$,
$A_4 : M_m.direction = M_n.direction$,
$A_5 : \{S_m, S_n\} \subseteq G$,
$A_6 : \{S_{m+1}, S_{n+1}\} \subseteq G'$,

where TS_i is a trail of user i, S and M represent the S and the M states, respectively, and G and G' are groups. Rule 1 is then formulated as follows:

$R_1 : A_1 \wedge A_2 \wedge A_3 \wedge A_4 \wedge A_5 \rightarrow A_6$.

For example, users A and B first encounter each other on the ground floor (user A is in state S_m and user B is in state S_n, S_m and S_n are in the same group based on the grouping algorithm). Later, each of them takes the elevator up at different time points. If they travel with the same distance, and end up in state S_{m+1} for user A and S_{n+1} for user B, we can infer that S_{m+1} and S_{n+1} are in the same group although they do not encounter on the new floor.

Following the same principle but performing a backward derivation, we have the second rule.

Rule 2. *Suppose people travel the same distance and in the same direction (i.e., up or down). If they end up on a floor and encounter each other, they must come from the same floor.*

Rule 2 is formulated as follows. $R_2 : A_1 \wedge A_2 \wedge A_3 \wedge A_4 \wedge A_6 \rightarrow A_5$.

By applying both Rule 1 and Rule 2, the groups generated by the grouping algorithm can be merged into larger groups. We do this by the merging algorithm.

After obtaining a number of groups, we sort all the groups by examining the state changes from one group to another. As a result, we obtain one or more directed graphs, in which each vertex represents a group and each directed edge represents the floor level information, pointing from a lower level to a higher level or a higher level to a lower level. If we only pick one type edge, pointing from a lower level to a higher level for example, the directed graph is basically a directed acyclic

graph (DAG). Finally, we select the longest path in the DAG, and the vertices in this path have a one-to-one correspondence to the floor levels. Thus the floor level of the groups and the S states in them are known, which means we know which floors the $M.NTrace$ is collected from.

Note that if there are more than one longest path, we can merge them based on the following rule. Since the groups in the longest paths have a one-to-one correspondence to the floor levels, the groups which correspond to the same floor level from different paths can be merged. As a result, we obtain a unique, longest path for the mapping table generation described in Subsection 3.6.

3.6 Mapping Table Generation

We are now ready to generate a table which maps the magnetic field signature to the floors. For each floor pair (i, j) in the final path we obtain, we first find the corresponding groups G_i and G_j for floors i and j, respectively. We then search for all segments, $\{S_i, M, S_j\}$, from the entire trails, where S_i is in G_i and S_j is in G_j, and get $Distance$ and $NTrace$ from M. Finally, we obtain a tuple $(FromFloor, ToFloor, Distance, MF\_Sig)$ for floor pair (i, j), where $FromFloor = i$ and $ToFloor = j$, $Distance$ is the altitude distance of floors i and j, and MF_Sig is $(NTrace_1, NTrace_2, ...)$, which contains magnetic field signatures of all ways (stairs, elevators and escalators) that a user can go from i to j.

This process runs incrementally with new trails until all the floor pairs are found, and a complete mapping table is built. For a building with f floors, the total number of floor pair (i, j) is $f(f-1)/2$. Actually, FTrack can locate users if only the mapping table contains the magnetic field signatures of all $(f-1)$ adjacent floor pairs, floor pairs $(i, i+1)$ for example. That is because the signature of floor pair (i, j) is actually the merge of floor pairs $(i, i+1), \ldots, (j-1, j)$. We show the mapping table of a 5-floor building in Fig.9, and the magnetometer traces are shown in Fig.10.

3.7 Locating a User

In the second phase, to locate a user using FTrack, mobile clients download the mapping table from the server. When the user travels in the building, the floor change activity can be detected and the magnetic field signature can be extracted using the mobile phone's barometer. By looking up the table, FTrack can track users' floor levels.

Floor Pair	Distance (m)	Magnetic Field Signature
1∼2	5	MF1 MF2 MF3 MF4
2∼3	4	MF1 MF2 MF3 MF4
3∼4	4	MF1 MF2 MF3 MF4
4∼5	4	MF1 MF2 MF3 MF4
1∼3	9	MF1 MF2 MF3 MF4
2∼4	8	MF1 MF2 MF3 MF4
⋮	⋮	⋮

Fig.9. Mapping table of a 5-floor building.

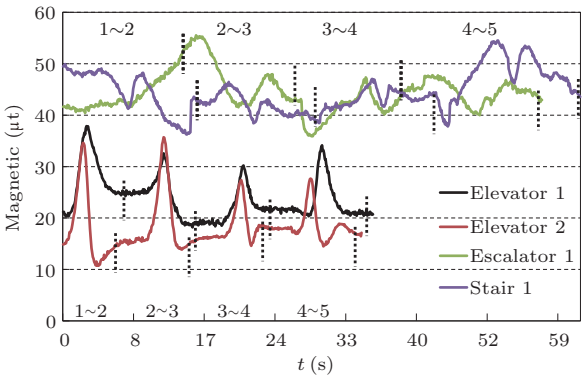


Fig.10. Magnetometer readings for different ways of going from floor 1 to floor 5.

Fig.11 shows the magnetometer traces collected when users go from floor 1 to floor 5 in the same stairwell. We can see that the fluctuation of the waveforms shows similar patterns. The data length is different and the waveforms are observed as a shift. This is because the time cost for different users to climb stairs has little difference based on their speed. Based on pattern matching with the magnetometer traces in the mapping table, we can find which floors the user is moving between. This approach is shown as follows.

3.7.1 *Feature Extraction*

First, the raw magnetometer readings may contain noise. If a data point value has apparent spark noise, it will be removed. After removing the noise, we smooth the readings with a window of 10. In order to compare the two traces, a straightforward approach is to use the

absolute value as the feature and calculate the mean squared error (MSE) of the two waveforms. Since the users' phones have not been calibrated (i.e., the readings of the two phones are different at the same place), there exists an unknown constant drift. This will cause errors when calculating the MSE value. In our approach, we use the variance as a feature to capture the fluctuation change instead of raw fingerprint values to avoid calibrating different magnetometers. For magnetometer traces, we obtain the variance as follows.

$$VarM(t) = M(t + \Delta t).m - M(t).m.$$

For example, Fig.12 shows the variance of the two magnetometer data traces of Fig.11.

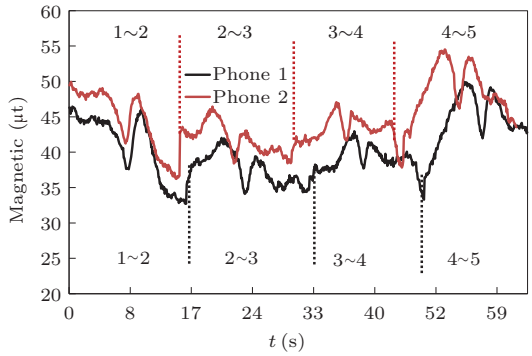


Fig.11. Magnetometer readings for two trips from floor 1 to floor 5 in the same stairs.

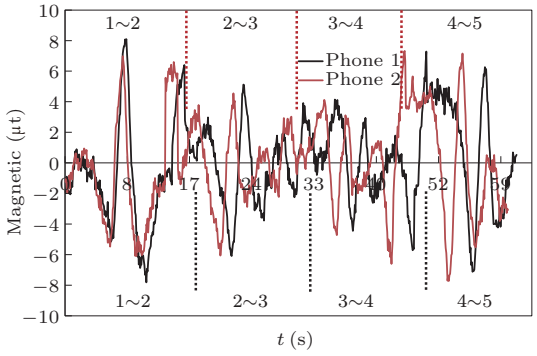


Fig.12. Variance feature of the magnetometer readings.

3.7.2 *Comparing Two Traces with DTW*

Since the sampling rate of different smartphones may have little difference, a pair of variation traces we collect between the same floors will have different lengths of data, which cannot be handled by MSE. In our approach, we apply the dynamic time warping distance measure (DTW)[20] which is less sensitive to the data length shift. To calculate the DTW, we first align the two variation traces. For example, for two time series of magnetometer variance $VarM_s$ and $VarM_l$, where

$$VarM_s = s_1, s_2, s_3, s_4, \ldots, s_n,$$
$$VarM_l = l_1, l_2, l_3, l_4, \ldots, l_m,$$

sequences $VarM_s$ and $VarM_l$ can be arranged to form an n-by-m plane or grid, as shown in Fig.13, where each grid $point(i, j)$ corresponds to an alignment between elements s_i and l_j. A warping path, W, maps or aligns the elements of $VarM_s$ and $VarM_l$.

$$W = w_1, w_2, w_3, w_4, \ldots, w_k.$$

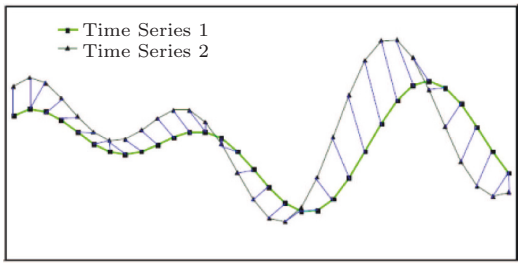


Fig.13. Dynamic time warping[20].

The dynamic time warping distance between two time series $VarM_s$ and $VarM_l$ is then:

$$\begin{aligned} &DTW(VarM_s, VarM_l) \\ &= \partial(First(VarM_s), First(VarM_l)) + \\ &\quad \min\{DTW(VarM_s, Rest(VarM_l)), \\ &\qquad DTW(VarM_l, Rest(VarM_s)), \\ &\qquad DTW(Rest(VarM_s), Rest(VarM_l))\}, \end{aligned} \quad (1)$$

where $First(x)$ is the first element of x, $Rest(x)$ is the remainder of the time series after $First(x)$ has been removed, and $\partial(i, j) = (s_i - l_j)^2$. From the DTW value in (1), we get the numeric measure of the similarity between two user variation traces.

In Subsection 3.6, we generate the mapping table. For a floor pair (i, j), we get a group of NTraces in MF_Sig, which contains magnetic field signatures of all ways (stairs, elevators and escalators) that a user can go from i to j. There may exist redundant NTraces in an MF_Sig, for example, there may exist three NTraces of the same elevator from i to j. In order to remove the redundant NTraces, we cluster NTraces by their DTW distance and remove the redundant ones.

3.7.3 Comparison with Mapping Table

When the user travels in the building, the floor change activity can be detected and the magnetic field signature can be extracted using the mobile phone's barometer. From the floor change activity detection result, we get the height change of the user Δh. Then we pick up the possible traces from the mapping table based on Δh. For example, in the mapping table of Fig.10, if a user went from floor 5 to floor 3, the floor change activity detection result shows the user went down with a distance of 8 m. From the mapping table, we can get two possible pairs of floors and eight possible traces based on 8 m. The floor pairs are $(5, 3)$ and $(4, 2)$, and there are eight traces because there exist four different ways for a user to change the floors (by elevators 1 and 2, stairs or escalator). Note that the magnetometer traces of floor pairs $(5, 3)$ are the reverse of floor pairs $(3, 5)$. If the signature of $(3, 5)$ is not directly contained in the mapping table, it is gotten by the merger of $(3, 4)$ and $(4, 5)$.

Here we get the user magnetometer trace and the possible traces (8 in this example) from the mapping table. We calculate the DTW distance between the user trace and the selected possible traces to find the one with the minimum distance. We conclude the floor pair of the selected trace is the floor pair of the user traveled. Thus the user's floor level is known.

3.8 Design for Robustness

3.8.1 Indoor/Outdoor Detection

For indoor/outdoor detection, we hope that every time we can locate the user into the right building. However, our recognition approach is not perfect, resulting in some false detections. For example, in the center of the city where there are buildings with high density, our hybrid indoor/outdoor detection approach may locate the user into the wrong building (i.e., a nearby building). In this case, we discuss the effectiveness in each phase of FTrack separately.

In the first phase of FTrack, if a user is wrongly located into another building, as showed in Fig.14, user e in building B is wrongly located to building A by FTrack. When FTrack generates the mapping table of building A using the encounter information, since user e is actually in building B, he/she will never encounter with any user in building A, and this fault case will not affect the generation of the mapping table. In building B, when FTrack finds user g encountered with user e in another building, FTrack will discard this encounter information. As a conclusion, when a user is wrongly located to another building, it will not affect the mapping table generation in any of the two buildings.

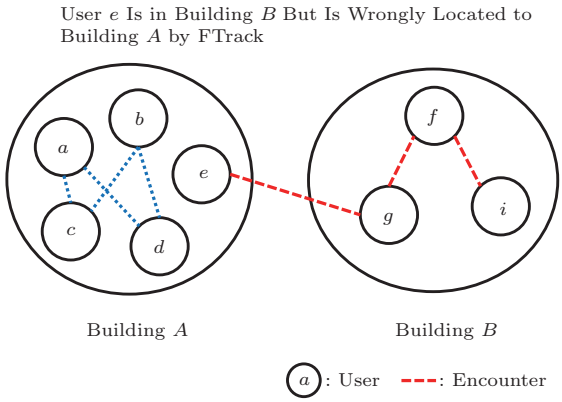


Fig.14. Indoor/outdoor detection fault analysis.

In the second phase, the building is used as an additional information when FTrack uses the mapping table to locate a user. When FTrack locates the user to the wrong building, we can infer that the floor information must be invalid. The conclusion is that whether FTrack locates the user into the right building is mainly determined by whether FTrack can locate the user to the correct floor. It does not affect the accuracy but is determined by the accuracy of FTrack.

3.8.2 Fault Analysis

As described above, FTrack relies on the detection of user activity and user encounter to operate correctly. However, the existing sensing and recognition techniques are not perfect, resulting in miss or false detections. In this subsection, we introduce the design for robustness, considering various fault cases.

For user activity detection, there are three cases of false detections:

F_1: an activity occurs, but it is not recognized by FTrack;

F_2: an activity is recognized by FTrack, but it actually does not occur;

F_3: an activity occurs, but it is recognized as a false activity or with wrong duration.

F_1 and F_3 are false positives, and F_2 is false negative.

For user encounter detection, there are two fault cases:

F_4: FTrack detects an encounter between two users, but they are located at different floors, e.g., typically adjacent floors;

F_5: an encounter occurs, but it is not detected.

F_4 is false negative and F_5 is false positive.

We use Fig.15 to illustrate the above fault cases. Fig.15(e) shows an example graph, resulting from a fault case, which all the S states on floors 2 and 3 are put into the same group (note that users in the same group who are in the S states should come from the same floor). This inconsistency could be caused by one of the four aforementioned false cases (i.e., $F_1 \sim F_4$). Fig.15(a) shows the correct graph where user 3 goes up from floor 2 to floor 3 where he/she encounters with user 1. However, in the case of F_1 where state G is not recognized, states C and F will be grouped together, resulting in the fault graph shown in Fig.15(e). Fig.15(b) shows the correct graph when user 1 encounters with user 3 on floor 2. If case F_2 occurs (i.e., state C is recognized, but it actually does not occur), states C and F will be grouped together, resulting in the same graph shown in Fig.15(e). Fig.15(c) also shows the correct graph when user 3 goes up from floor 1 to floor 3 and encounters with user 1. If F_3 occurs (i.e., state F is not correctly recognized or the duration from state G to state F is wrong), states F and E will be grouped into the same group. Fig.15(d) shows the correct graph when no encounter occurs between user 1 in state C and user 3 in state F. However, in the case of F_4 where an encounter between them is falsely detected, states C and F will be grouped together, resulting in the same graph shown in Fig.15(e).

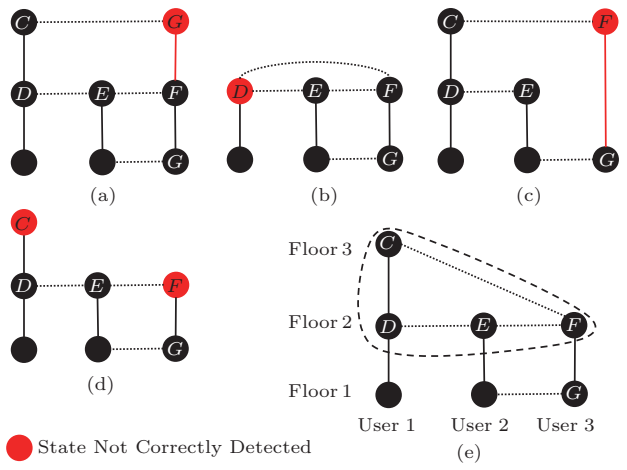


Fig.15. Cases illustration. (a) Fault case F_1. (b) Fault case F_2. (c) Fault case F_3. (d) Fault case F_4. (e) Resulting graph from inconsistency.

Fault cases $F_1 \sim F_4$ may cause inconsistency leading malfunction in FTrack. Detecting such inconsistency is important to the robustness design of FTrack. Note that we do not handle F_5 here as it has been taken care of in both grouping and merging algorithms, as explained in Subsection 3.3.

3.8.3 Inconsistency Detection

Detecting inconsistency is done every time when a new group is formed by the grouping or merging algorithm. FTrack detects inconsistency by checking whether all S states in the same trail segment are in different groups. If there exist two S states in the same group (e.g., states C and D of user 1 shown in Fig.15(e)), a fault case must occur because the S states in the same trail segment must be on different floors, and the S states on different floors should not be grouped into the same group. Hence, we have the rule as follows.

Rule 3. *For the S states in a trail segment, if two or more of them are put into the same group, an inconsistency is found.*

To formulate the rule, we define the following propositions.

$A_6 : \exists\{S_m, M_m, \ldots, S_n\} \subseteq TS$,

$A_7 : \{S_m, S_n\} \subseteq G$,

A_8 : there exists inconsistency in G,

where TS is a trail segment, S and M represent the S and the M states, respectively, and G is a group. Rule 3 is then formulated as follows: $R_3 : A_6 \wedge A_7 \rightarrow A_8$.

3.8.4 Removing Inconsistency

Removing inconsistency is also done during the processes of grouping and merging. We use a more complicated example shown in Fig.16 to illustrate the steps of removing inconsistency. We first find all inconsistent pairs of S states (e.g., (A, B) and (C, D)). We then find the shortest path between the two S states in each pair, and they are $Path\_AB = \{A, C, F, D, B\}$ and $Path\_CD = \{C, F, D\}$. There must exist a path between them because they are in the same group. We compare the two path lengths, and select the shortest one, i.e., $Path\_CD$. We remove the encounter relationship between the S states in this path (e.g., encounters CF and DF of $Path\_CD$), and finally, we obtain a new graph, as shown in Fig.17.

All the removed paths are stored during the process. If an S state or two adjacent S states in the same trail segment occur more than once in the removed paths, the nodes will be removed from the trail segment and

the trail segment will be divided into two trial segments. We discover the shortest path because it contains the minimum number of the S states. Since there must exist a fault in this path, removing their encounter relationship will remove the inconsistency. If fault case F_1 or F_2 occurs, the wrongly detected S states often cause more than one inconsistency, because every encounter after the wrong detection is a fault encounter of F_4. Therefore, if an S state or two adjacent S states appear more than once in the paths to be removed, we remove them.

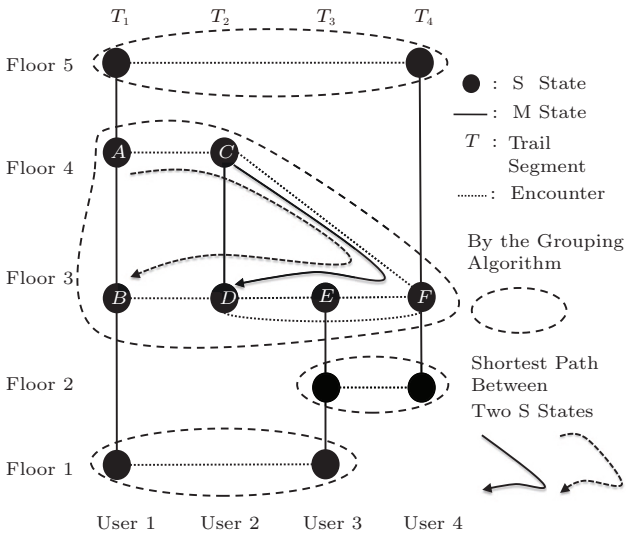


Fig.16. Fault example.

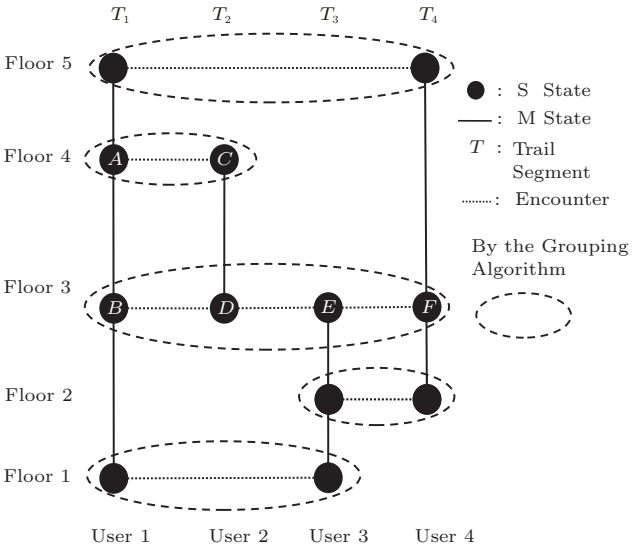


Fig.17. Fault removed.

4 Theoretical Analysis

We provide the theoretical analysis in this section.

4.1 Modeling of Groups

We model the S states using a random graph $\Gamma_{n,N}$ with n labeled vertices and N edges. A group can be viewed as a set of connected vertices (i.e., connected component[21]) in $\Gamma_{n,N}$, where a vertex represents an S state in the group (n is the total number of S states), and an edge may exist between two S states according to our grouping and merging algorithms.

4.2 Finding p'

We denote p as the probability that there exists an edge between any two vertices in $\Gamma_{n,N}$ which is created only by the grouping algorithm (i.e., user encounters), and we assume an equal p for people encountering each other on any floor. We denote p' as the probability that there exists an edge between any two vertices which is created by both the grouping and the merging algorithms (i.e., user encounters and merging rules).

We now compute the probability of p'_k for any floor, i.e., floor k. p'_k consists of p (i.e., the probability of people encounter on floor k) and p'_j ($j \neq k$) from any other floor j. For any pair of the S states on floor k, the probability that their next S states are on floor m ($k < m$) is $1/(f-k)^2$. We know the probability of merging any two S states on floor m is p'_m. Therefore, the probability of merging any pair of S states on floor k by the S states on all the upper floors is $(p'_{k+1} + p'_{k+2} \ldots + p'_f)/(f-k)^2$, and by the S states on all the lower floor levels is $(p'_1 + p'_2 \ldots + p'_{k-1})/(k-1)^2$. We then obtain p' on floor k as follows.

$$p'_k = \begin{cases} p + \dfrac{(p'_{k+1} + p'_{k+2} + \ldots + p'_f)}{(f-k)^2}, & \text{if } k = 1, \\ p + \dfrac{(p'_1 + p'_2 + \ldots + p'_{k-1})}{(k-1)^2} + \\ \dfrac{(p'_{k+1} + p'_{k+2} + \ldots + p'_f)}{(f-k)^2}, & \text{if } k \in (1, f), \\ p + \dfrac{(p'_1 + p'_2 + \ldots + p'_{k-1})}{(k-1)^2}, & \text{if } k = f, \end{cases} \tag{2}$$

where f is the total number of floors. We plot the ratio of p' vs p in Fig.18 for different floor levels (i.e., 5 levels to 40 levels), where each p'_k is set to p initially. The figure shows p' converges after iterations, and it also shows a bigger ratio of p' vs p for lower buildings.

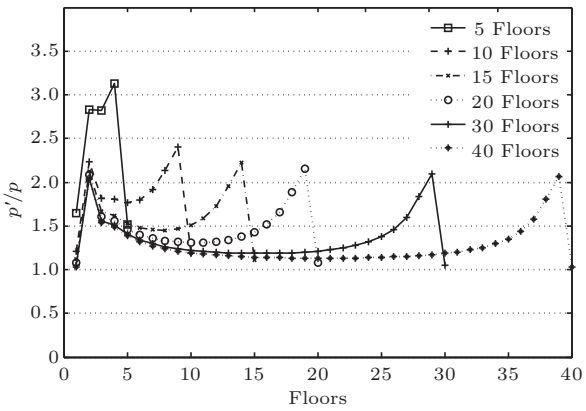


Fig.18. p' vs p.

4.3 Proofs of Completeness and Correctness

It is shown in [21] that the random graph mentioned in Subsection 4.1 is almost surely connected if

$$p' > \frac{1}{n}(1+\epsilon)\ln n.$$

With a large n, the value of $((1+\epsilon)\ln n)/n$ is small, and therefore a smaller p' can meet the above equation. It implies that with enough S states, after grouping and merging, there exists a large group on each floor which may contain all the S states on this floor. We denote the largest group on floor i as G_i.

Theorem 1. *For any G_i and G_{i+1}, if our algorithm can infer which one is on a higher floor, we can then surely find a path which each group in the path has a one-to-one correspondence to the floor level.*

Proof. Since our algorithm knows which group (G_i or G_{i+1}) is higher, after sorting all the groups, there must exist a path from G_1 to G_f, which has a one-to-one correspondence to the floor level. □

With Theorem 1, we define the correctness of our algorithm as correctly finding the number of floors.

Theorem 2. *For any G_i and G_j, if our algorithm can infer which one is on a higher floor, we can then surely find all the floor pairs (i, j), where $i < j$.*

Proof. Since our algorithm knows which group (G_i or G_j) is higher, by Theorem 1, we can find the floor levels of each group. Furthermore, there exists a segment $\{S_i, M_i, S_j\}$ with S_i in G_i and S_j in G_j. Based on M_i, we can find the floor pair (i, j). Finally, all the floor pairs can be found. □

With Theorem 2, we define the completeness of our algorithm as correctly finding all the floor pairs.

We now analyze the correctness and the completeness of our algorithm. We define $P_{i,j}$ as the probability our algorithm can find which group (G_i or G_j) is higher. We denote n'_i and n_i as the number of the S states in G_i and on floor i, respectively.

We show how to find $P_{i,j}(i < j)$. For any S state, S_k, on floor k, its preceding state in the trail segment is denoted as $S_{k,\mathrm{pre}}$. If there exists an S' in G_j, and its preceding state is in G_i, the algorithm can infer that G_j is on a higher floor than G_i. Therefore, $P_{i,j}$ is equal to the probability that at least one S' exists. For any S_j in G_j, we first get the probability that $S_{j,\mathrm{pre}}$ is on floor i, which is $1/(j-1)$. Then, if $S_{j,\mathrm{pre}}$ is on floor i, we can get the probability that it is not in G_i, $1 - n'_i/n_i$. Therefore, the probability that at least one S' exists is shown in (3).

$$P_{i,j}(i<j) = 1 - \left(1 - \frac{n'_i}{n_i}\right)^{\frac{n'_j}{j-1}}. \tag{3}$$

Finally, our algorithm is correct with a probability of

$$P = \prod_{2}^{f} P_{i-1,i},$$

and our algorithm is complete with a probability of

$$P_{\mathrm{all}} = \prod_{1 \leqslant i < j}^{f} P_{i,j}.$$

From (2), we have $n'/n \approx 1$, and $P_{i,j}$ is high with a large n. When $n \to \infty$, we then have $P = 1$ and $P_{\mathrm{all}} = 1$. With enough S states collected, in a high probability (≈ 1), we are able to find the correct number of floor levels and all the floor pairs.

5 Evaluation

We evaluate FTrack using simulation and field studies. In this section, we first present simulation results, and then report the experiments from the field study.

5.1 Simulation Methodology

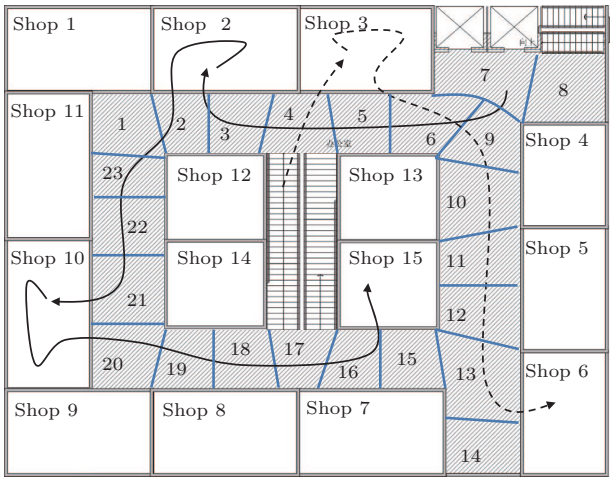
We design a simulator to evaluate the efficiency, scalability and robustness of FTrack. We simulate the processes of taking the elevator/escalator and walking on the stairs in two scenarios — an office building and a shopping mall. The simulation consists of the following two processes. The first process simulates the vertical mobility model which defines how users go up and down in the building (i.e., taking the elevator/escalator or walking on stairs). In addition, we also introduce a horizontal mobility model which defines how users move on each floor (i.e., how they encounter each other).

5.1.1 Vertical Mobility Model

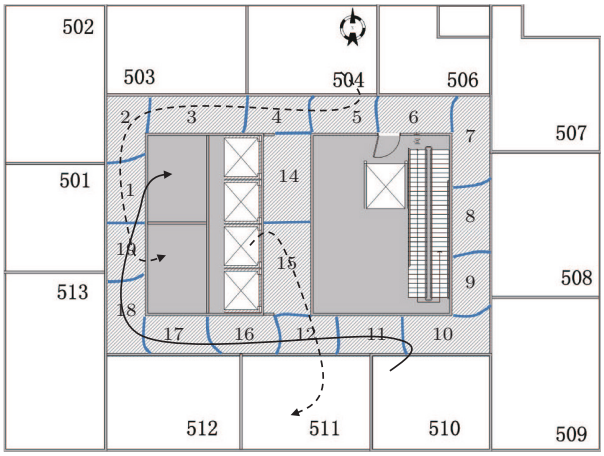
In the vertical mobility model, we simulate the processes of taking the elevator/escalator and walking on stairs. The simulation process of taking the elevator is divided into cycles, and each cycle simulates the process that the elevator goes up from, or down to the ground floor, with people entering and leaving the elevator from or to any levels. We model the process of people entering the elevator from the ground floor as the Poisson distribution. The expected number of the Poisson distribution is set to half of the maximum load of a typical elevator (i.e., eight persons). People on the ground floor may go up to any floor with a probability of $1/(f-1)$, where f is the number of floors of the building. From any other floor, a person may enter the elevator with a probability of p_e, and go to other $(f-2)$ floors with an equal probability $1/(f-2+k)$, where k is a constant, except to the ground floor which is k times bigger (i.e., $k/(f-2+k)$). When people enter or exit the elevator from a floor, the number of people on that floor gets updated. By multiplying p_e, we obtain the number of persons entering the elevator. Each cycle starts from the ground floor. We first compute the number of people entering the elevator and which floor they want to go, and the elevator goes up from the ground floor, stops when people exit or enter the elevator, and comes back the ground floor when the cycle ends. We assume five minutes for each cycle on average, i.e., about 100 users per hour. User trails are collected continuously over the time. User encounters in the elevator are also recorded. Simulating the process of taking escalators or walking on stairs is rather simple: each user is given a random destination floor, and then he/she will go there by taking the escalator or walking on stairs.

5.1.2 Horizontal Mobility Model

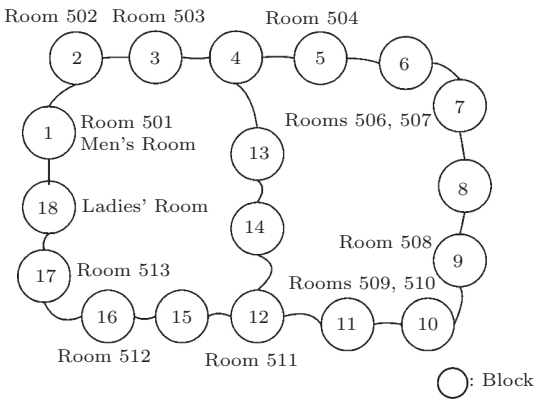
To provide encounter information, we use two approaches. The first one is based on a probability which indicates how often users encounter each other in the building. We set the probability to 0.01, although the value in a real scenario may be much larger, and we set a fixed probability for all the users for simplicity. In the second approach, inspired by [22], we propose a horizontal mobility model to simulate user movement on the same floor. This model is more realistic and complex. We describe the detail for both an office building and a shopping mall environment as follows. The horizontal mobility model relies on a floor plan which typically consists of rooms or shops (they are the destinations) and corridors. For example, Fig.19(b) shows a floor plan of a local office building. We first divide



(a)



(b)



(c)

Fig.19. Floor plan examples (the solid and the dotted lines with arrows indicate user moving traces). (a) In shopping mall. (b) In office building. (c) Graph of blocks.

the entire corridor area into blocks whose size is limited by the range of our encounter detection (i.e., typically 2∼3 meters). If two users appear in the same block, the model defines that they encounter with each other. These blocks are then interconnected to form a graph which is shown in Fig.19(c). A node in the graph represents a block, corresponding to some user destinations (e.g., rooms or shops). Each user is assigned with an initial block which is near to the entrance of the elevator/escalator, and the staircases. User walking speed and the time duration staying in the blocks follow the Poisson distribution. To generate the movement trace of a user in the simulation, we set the parameters as shown in Table 2.

Fig.19(a) shows one of the floor plans in the shopping mall. There are 15 shops and one corridor, and the corridor is divided into 23 blocks. Users arrive or leave this floor by escalators or elevators through blocks 4, 7, 8 and 17. For each user, we first randomly choose the shops to visit based on the popularity of each shop. The number of shops a user visits follows the Poisson distribution. The expected number of the Poisson distribution is set to half of the number of shops. We randomly assign the popularity for each shop while in reality it can be computed based on the number of visitors. We then generate the moving trace of the user which is shown in Fig.19(a).

To simulate the process of user walking on this floor, we set the walking speed to 0.8 m/s[23], which is slower than the average human walking speed (i.e., 1.4 m/s). The time duration staying in each shop is modeled based on the Poisson distribution with the expected number of 1 minute and 10 minutes for window and non-window shoppers, respectively. For users in the same shop, we simulate their encounters with a probability p, where p can be different from one shop to another. All the encounters of users in a block or a shop will be recorded. Similar settings are applied to the office building as shown in Fig.19(b).

FTrack computes the mapping table using the user trails generated and the encounter information recorded, which include 5% of noise. The process runs recursively, and we report our results in the next subsection.

5.2 Simulation Results

Figs.20(a) and 20(d) show the CDF when the correct number of floors is found by both the grouping and merging algorithms, and the grouping algorithm only, respectively. While the results show that it requires less time with fewer floors for both cases, with the same number of floors, the combination of grouping and merging algorithms computes the correct results much faster. One observation is that for a tall building (i.e., 20-floor in the figures), the CDF for the combination of grouping and merging algorithms grows steeply, demonstrating the effect of the merging algorithm. Fig.20(a) also shows that our algorithms find the correct number of floors in less than four hours in a 10-floor building, and less than 10 hours in a 20-floor building. Note that Figs.20(a) and 20(d) are the results with a fixed probability of user encounters, and the mobility model is not used. In Fig.21(a), we show the CDF for both the shopping mall and the office building with the horizontal mobility model. Fig.21(a) shows that the correct number of floors is found in less than 1.5 hours in a 10-floor building, and less than 5 hours in a 20-floor office building. The time taken to find the correct number of floors with the horizontal mobility model is much less than that without this model, and it clearly demonstrates the potential and efficiency of FTrack when applied in a real scenario. One observation is that the time taken to find the correct number of floors in the shopping mall is less than that in the office building with the same floor levels.

Fig.20(b) shows the number of groups computed over time for different building cases. We observe a similar trend for all the cases. In the beginning, the number of groups grows since more people travel between different levels (i.e., more encounters). When more groups are merged, the number of groups decreases and converges to a value which is equal to the maximum number of floors for each case.

Table 2. Simulation Parameter Settings

	Number of Destinations	Number of of Blocks	Initial Blocks	Moving Speed by Poisson (m/s)	Number of Destinations to Visit	Time Duration Stay in Destinations by Poisson (min)	Encounter Probability in Destinations
Shopping mall	15	23	4, 7, 8, 17	Avg=0.8	Avg=7	Avg=1, 10	p
Office building	12	19	14, 15	Avg=1.4	1 room, 1 toilet	Avg=120, 2	50%

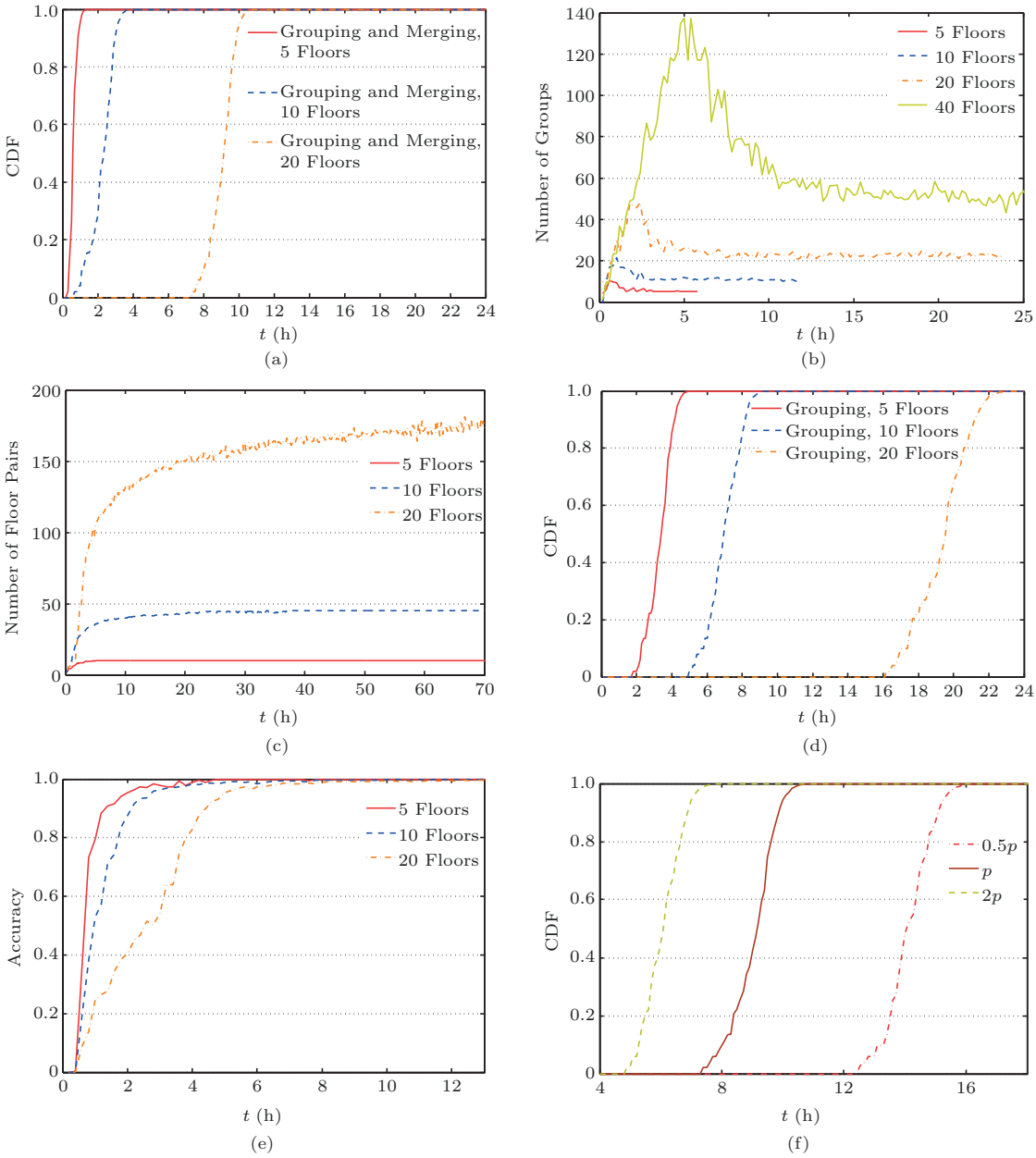


Fig.20. Simulation results with a fixed probability of user encounters. (a) CDF when the correct number of floors is found by both the grouping and the merging algorithms. (b) Number of groups. (c) Number of floor pairs. (d) CDF when the correct number of floors is found by the grouping algorithm. (e) Accuracy of FTrack. (f) CDF when the correct number of floors is found ($f = 20, p = 0.01$).

Figs.20(c) and 21(b) show the time spent by our algorithms on creating the complete mapping table with a fixed user encounter probability and the horizontal mobility model, respectively. The maximum number of tuples (i.e., floor pairs) for a building with f floors can be calculated by $f(f-1)/2$. For example, we have 10, 45 and 180 tuples for 5-, 10-, and 20-floor buildings, respectively. With enough time, we can obtain all the tuples. Higher buildings require much longer time because 1) more tuples will be computed, and 2) some tuples may be rare (e.g., the elevator rarely goes up from the ground floor to the 20th floor without a stop in the middle). Fig.20(c) also shows that our algorithms are capable of computing most of the tuples quickly.

Figs.20(e) and 21(c) show the accuracy of FTrack with a fixed user encounter probability and the horizontal mobility model, respectively. Both figures show that the accuracy increases with more tuples computed. For example, FTrack achieves an accuracy of over 90%

after 2.2 hours in our simulation. When the complete set of tuples is obtained, the accuracy achieves near 100%. While the merging algorithm is effective, its efficiency is significantly influenced by user encounters. We run the merging algorithm with different encounter probabilities and the result is shown in Fig.20(f), where t is the time when the correct number of floors is found in a 20-floor building. The figure shows that the result is obtained faster with a higher probability of user encounters.

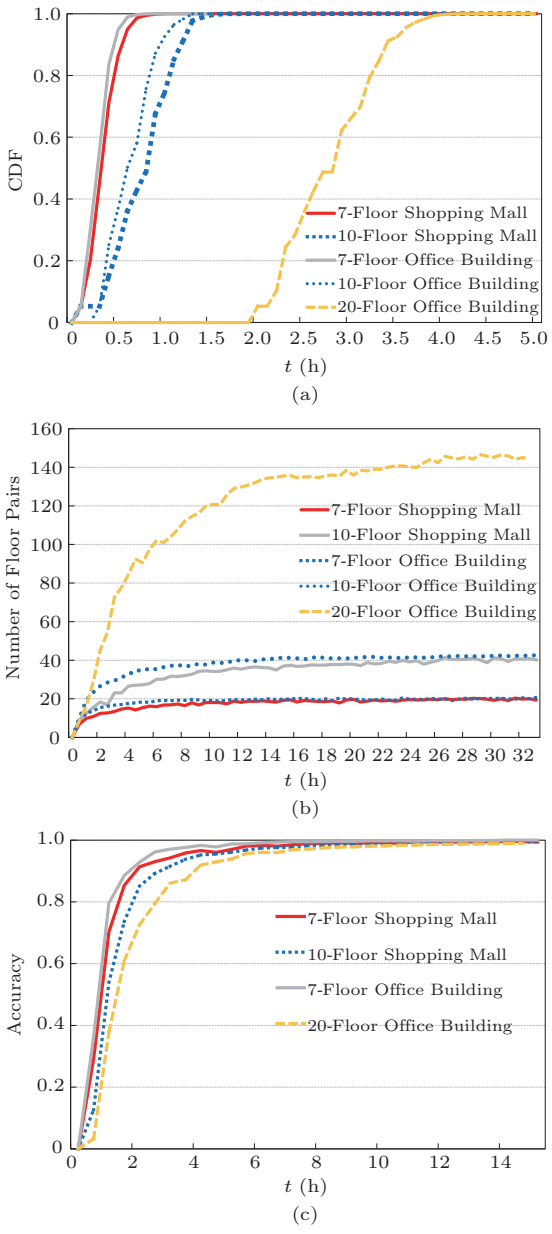


Fig.21. Simulation results with horizontal mobility model. (a) CDF when the correct number of floors is found. (b) Number of floor pairs. (c) Accuracy of FTrack.

5.3 Field Study

Our simulation results show that the performance of FTrack is influenced by the probability of user encounters. We are interested in how FTrack behaves under a real-world encounter probability. We conduct a field study which involves 15 mobile users in three different buildings as shown in Fig.22, including 10 university students, three professors, one cleaner, and one security guard. Among them, three are females and 12 are males, and all aged between 20 and 40. The information of the buildings is shown in Table 3. As an example, the floor layout of the office building is shown in Fig.23. We use different Android smartphones (e.g., Samsung, MOTO and HTC). Each smartphone is equipped with an embedded barometer and a mobile network data connection (i.e., GPRS or 3G). The field study is conducted as follows. We implement a prototype system and publish it on the website③. The users all install it on their smartphones. In every building, users are told to walk and change floors randomly, and each user can choose either taking the elevator/escalator or walking on the stairs as he/she wish. For example, in the office building, following their daily routines, student volunteers usually go to their offices for work in the morning and leave in the evening. They may go out for lunch/dinner, or attend classes in other buildings in the morning or afternoon. They may go to other floors, for example, playing table tennis at floor 7, collecting postal mails/articles at floor 2, attending meetings and meeting with their supervisors at floors 3∼10. The cleaner performs cleaning tasks at certain floors each day. The safety guard stays mostly at the ground floor, but he/she may occasionally patrol some random floors at any time of a day.

(a) (b) (c)

Fig.22. Buildings in the field study. (a) Shopping mall. (b) Office building. (c) Hotel building.

③Application and code. moon.nju.edu.cn/twiki/bin/view/ICSatNJU, May 2015.

Table 3. Buildings Information in Field Study

Building Type	Floor Levels	Floor Height (m)	Number of Elevators	Number of Stairs	Number of Escalators
Shopping mall	5	6	5	3	4
Office building	10	5.8 (f1, f2), 4	3	2	0
Hotel building	20	3.6	5	2	1 (f1∼f2)

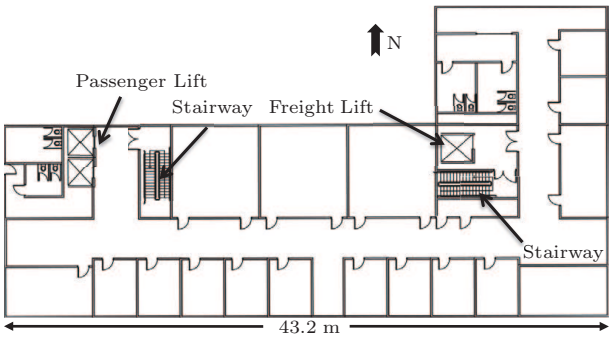


Fig.23. Floor layout of the office building.

When the application runs, the sampling rate is one sample every 10 seconds for GPS, one sample per second for light sensor and five samples per second for magnetometer sensor. When the user is detected to be in a building, the GPS and the light sensor will stop sensing. At the same time the client will collect barometer readings at a rate of two samples per second, collect magnetometer readings five samples per second, and get a Bluetooth sample every five seconds, and all the samples will be logged in a data file which will be uploaded to the cloud server every 10 minutes. The floor change activity detection is done in real time and the user trail will also be uploaded to the cloud server. The client also performs time synchronization with the server by computing the round-trip delay time and the offset. If the mapping table is available in the cloud server, it will be downloaded to locate the user. To get the ground truth, every user takes an audio recorder and is told to record the information of floor levels and the ways of changing floors when he/she changes a floor. The field study runs for 2/3/5 hours in the shopping/office/hotel building respectively. In each mobile client, the history of floor location is logged. In the cloud server, all the uploaded data and the generated mapping table are logged. We analyze the performance based on the logged data in both the client and the cloud server.

5.4 Field Study Results

Fig.24(a) shows that in the office building, FTrack detects the correct number of floors after about 70 minutes when the complete trails are obtained, indicated by the point of correctness. The time required to detect the correct number of floors is less than that in our simulation due to that more encounters occur in our field study, especially in the elevator. The figure

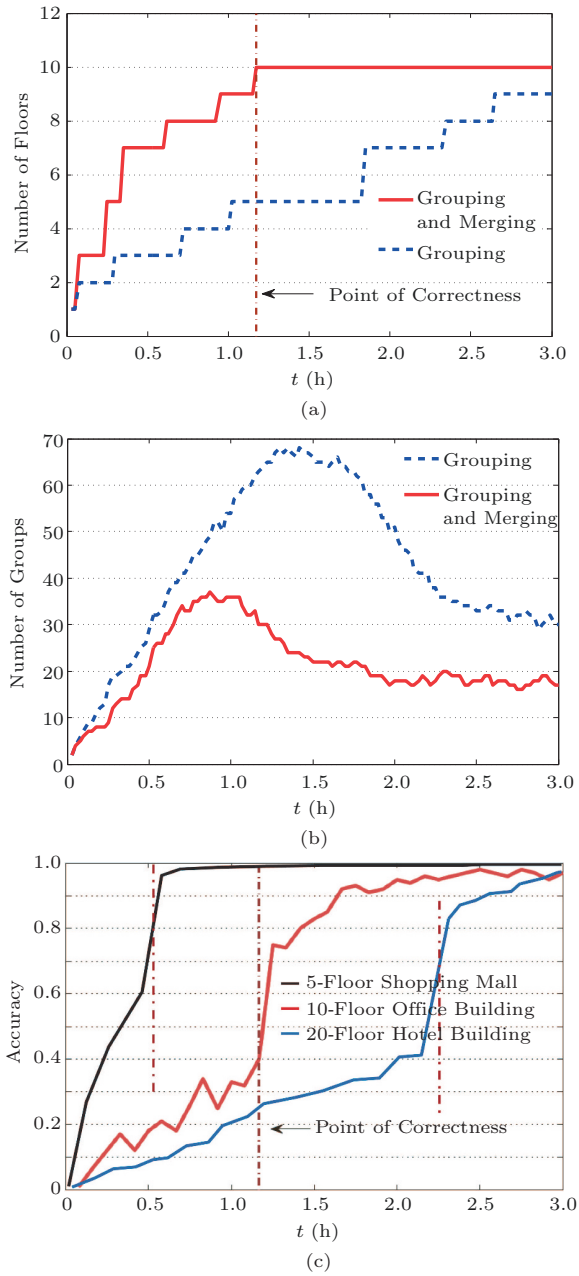


Fig.24. Field study results in the office building. (a) Number of floors detected. (b) Number of groups. (c) Accuracy of FTrack.

also shows that by the merging algorithm, FTrack obtains the complete trails more effectively than the grouping algorithm.

Fig.24(b) shows the number of groups generated over time. While we observe a similar trend to that seen in our simulation, the number of groups in the field study takes longer time to converge compared with that in our simulation. One explanation is that we observe that users in our field study do not stay on a floor for a long time, and they do not encounter each other more often in this special situation, resulting in many individual, discrete groups which require a longer time to merge.

Fig.24(c) shows the accuracy of FTrack. At the point of correctness, the accuracy increases steeply and faster in lower buildings. The other observation is that in the office building, the accuracy approaches 92% in two hours, and reaches 97% in three hours. All buildings can get an accuracy of 96% in about three hours.

5.5 Energy Consumption

In the first phase, FTrack uses crowdsourcing-based approach to generate the mapping table of every building. In this phase, FTrack client in the smartphone keeps GPS, light sensor and magnetometer sensor on until the user goes into the building. The sampling rate is one sample every 10 seconds for GPS, one sample per second for light sensor and five samples per second for magnetometer sensor. When the user is moving in the building, the barometer, magnetometer and Bluetooth should be kept on all the time. The sampling rate is 2/5 samples per second for barometer and magnetometer sensor, one sample every five seconds for the Bluetooth. Besides the energy consumption of sensor sampling and data computing, FTrack client will consume some energy when uploading the user trace to the cloud server every 10 minutes. This is all the energy consumption source of FTrack in the first phase. It is important to notice that the first phase to build the mapping table is once for all and can be finished in a short time (five hours in our field study), and thus this energy consumption is acceptable.

To verify the real energy consumption, we evaluate the energy consumption of the first phase using a Samsung Galaxy Nexus smartphone running Android 4.1 OS. The power consumption is computed based on PowerTutor, a diagnostic tool for analyzing system and application power usage from the Android market. The result shows that the mobile phone's battery level drops from 100% to 30% after running this experiment for six hours. The power consumption of our data collection application is about 225 mW (i.e., 51.6% of the total consumption in the standby mode).

In the second phase, FTrack is working in the normal mode, and it keeps on getting 2/5 samples per second for barometer and magnetometer sensor. When a floor change activity is detected, FTrack locates the user's floor level by looking up the previously downloaded mapping table. In this process, the energy is mainly consumed by the sampling of barometer and magnetometer sensor, which is minimal.

We sum up the energy consumption in different phases of FTrack in Table 4. Similar to other localization techniques, the first crowdsourcing phase (training phase) of FTrack consumes more energy, but we believe that this is acceptable because this phase is once for all and can be finished in a short time. In the second phase, FTrack is very energy efficient and makes the system practically useful.

5.6 Energy Consumption Optimization

In order to get even better energy consumption performance, we take advantage of some efficient approaches in FTrack. In the first phase, GPS and Blue-

Table 4. Energy Consumption Source of FTrack

Phase	Energy Consumption Source	Average Power Consumption (mW)	Note
First phase (training phase)	GPS: 10 seconds per sample Light: 1 sample per second Magnetometer: 5 samples per second Barometer: 2 samples per second Bluetooth: 5 seconds per sample Data computing Upload data	225	Once for all (within 5 hours)
Second phase (normal usage)	Magnetometer: 5 samples per second Barometer: 2 samples per second Calculate the location	32	Normal usage (happens most)

tooth cost most of the energy. An obvious way to reduce the energy consumption of GPS is to reduce the sampling rate, resulting in the low accuracy of in-building detection, which is unacceptable. In our approach, we try to save energy by turning on and off the GPS at the appropriate time, instead of keeping the GPS on. We only turn the GPS on when the light intensity is higher and the magnetic field variance is smaller than a threshold. Once a user is detected to be inside a building, the GPS will be turned off immediately. This approach reduces the GPS energy consumption dramatically. For the Bluetooth, we successfully reduce the power consumption by making use of the Bluetooth Low Energy (BLE) technology, which is designed to provide significantly lower power consumption compared with the traditional approach.

In the second phase, the highest energy consumption source is the sampling of barometer and magnetometer readings. The barometer sampling cannot stop because users may change floors at any time. For magnetometer readings, since we only need magnetic field data when the user is changing the floors, we change the sampling strategy to only start sampling when the user is changing floors.

5.7 Comparison

We compare FTrack with two typical solutions, 1) RADAR[6] which uses Wi-Fi fingerprints to locate the user, and 2) SkyLoc[24] which uses GSM fingerprints to address the floor localization problem. We compare them in the following aspects, 1) accuracy and precision, 2) energy efficiency, 3) scalability and cost, and 4) security and privacy. The accuracy and energy efficiency result is obtained by our application called "Bit movement at school", which is a campus service application which provides sport and study information to students based on their locations, and 28 students use this application. We implement three location services using the three approaches respectively and test their accuracy and energy performance. The results are summarized in Table 5.

Accuracy and Precision. We only measure the accuracy of locating the user to the right floor. Under the same conditions, the results are 96% by FTrack, 89% by RADAR, and 68% by SkyLoc, and the accuracy is slightly different from the results reported in the original papers, but they have no significant difference. Our experiment shows that FTrack outperforms Wi-Fi and GSM fingerprint based approach.

Energy Efficiency. Under the same situations, the results are FTrack 32 mW, RADAR 58 mW, and SkyLoc 62 mW. The sampling of barometer and magnetometer sensors consumes similar energy compared with sampling the Wi-Fi and GSM. RADAR and SkyLoc need to connect to the server for location calculation, while FTrack does this locally. In the training phase, which is once for all, the energy consumption performance is acceptable. RADAR and SkyLoc need the developer to war-drive every position of the building, and the energy consumption is concentrated in the developer's smartphone. FTrack uses crowdsourcing to generate the mapping table, and the energy consumption appears in every user who participates in the crowdsourcing process.

Scalability and Cost. RADAR and SkyLoc can only work in the buildings with full Wi-Fi or GSM infrastructure, which may not be available everywhere. In developing countries, many buildings have no or sparse Wi-Fi which is not dense enough for localization. Even in developed countries, studies show Wi-Fi may not be fully available in many buildings. FTrack has a better scalability because it does not rely on any infrastructure of the building. RADAR and SkyLoc need developers to war-drive every position of the building to collect the Wi-Fi and GSM fingerprints. This is both time-consuming and labor-intensive. On the contrary, FTrack is based on crowdsourcing, which means the process can be finished in a natural way. Nobody needs to do anything specifically.

Security and Privacy. RADAR and SkyLoc need a location server to calculate the location of the user, the security and privacy cannot be guaranteed if the server is attacked. In FTrack, the mapping table of a building is small and can be easily downloaded to the user's smartphone. The location process can be done locally on the smartphone, which has better security and privacy performance.

Table 5. Summary and Comparison of FTrack with Similar Work

Technology Name	Accuracy and Precision (%)	Energy Efficiency (mW)	Scalability and Cost	Security and Privacy
FTrack	96	32	Infrastructure free crowdsourcing	Good
RADAR	89	58	Need Wi-Fi and war-drive	Poor
SkyLoc	68	62	Need GSM and war-drive	Poor

6 Related Work

Existing techniques for localization rely on deployed radios (e.g., Wi-Fi access points, GSM base stations) and make different assumptions about infrastructure and calibration. RADAR[6] operates on Wi-Fi fingerprints, and is capable of achieving high accuracy in indoor deployments. However, RADAR needs to calibrate Wi-Fi signal strengths at many physical locations in a building. The calibration process is time-consuming and may not scale over larger areas. PlaceLab[5] uses Wi-Fi and GSM signals. A radio map is created by war-driving car-accessible roads and mapping the Wi-Fi access points or GSM-based stations to GPS coordinates. The radio map is distributed to mobile devices which localize themselves by comparing overheard access points or GSM-based stations with those recorded in the map. Some recent approaches such as LiFS[8] use crowdsourcing to reduce the war-driving cost to some extent, but this paper involves a complicated training process. Different from these systems, FTrack does not require any pre-installed infrastructure.

Sensor-assisted localization methods have been proposed with the popularity of smartphones. These systems typically make use of accelerometer and electronic compass on the smartphone. The basic idea is to use acceleration data to measure the walking speed, and electronic compass to compute the orientation of a user. To overcome inherent noise in sensor readings, Constandache *et al.*[9-10] proposed a better approach by identifying acceleration signatures in human walking patterns (i.e., the nature up and down bounce), and then multiplying step count with the user's step size to obtain the location. However, Escort[9] leverages fixed beacons for calibration, and CompAcc[10] makes use of possible walking paths extracted from Google Maps.

SkyLoc[24] uses a GSM fingerprinting-based approach to address the floor localization problem using mobile phones. It identifies the floor correctly in up to 73% of the cases and is within two floors in 95% of the cases. It leverages on GSM signals which may vary significantly in indoor environments. The training process in SkyLoc is time-consuming. In addition, the solution may not be scalable since training is required for each building.

Several recent approaches[8,13-14] use crowdsourcing to construct the floor map for localization; however, they rely on Wi-Fi infrastructure and require complicate training to achieve good accuracy. Paper [18] studies on the properties of mobile-embedded barometers across a number of buildings, but fails to solve the problem of using the barometer to determine the floor of a user. There was another solution proposed by Wang *et al.*[19] before the mobile-embedded barometers appeared. Wang *et al.* used a barometer sensor to track users' floor level. As we discussed in the introduction, this approach needs the detail information of the building and needs to know the initial floor of every user, which is hard to get. Furthermore, a miss or wrong detection will cause serious errors in the later localization. FTrack does not require any calibration and any prior knowledge of the building, and yet we achieve high accuracy. It is not a tracking system and it does not rely on the previous location information. FTrack detects user activities of changing floor by a novel barometer-based technique, and it has no assumption of users' walking pattern or the ways to carry/use mobile phones.

Our work is similar to a field in AI and robotics in some ways. The field is called simultaneous localization and mapping (SLAM). Authors of FastSLAM[25] proposed an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations, and scales logarithmically with the number of landmarks in the map. The way of building the map by the moving robot is similar to our approach of building the map of the building. However, there are still some differences. 1) Our approach of building the mapping table is based on crowdsourcing, which is done by the combination of information collected from different users. 2) Our mapping table is a magnetic field signature map. 3) Our approach does not need to deploy the infrastructure, and does not need any landmarks. Particle filter is a very important tool in this area. We have considered the use of particle filters to improve the performance of the system.

7 Conclusions

In conclusion, this paper presents a novel, scalable floor localization scheme. FTrack uses the mobile phone's sensors only, and it requires neither any infrastructure nor any prior knowledge of the building. Through crowdsourcing, FTrack is able to find the magnetic field signature between floors, which can be then used by mobile users to pinpoint their current floor levels. We analyzed different fault cases and designed an algorithm to improve the robustness of FTrack. Through our simulation and field studies, we demonstrated the efficiency, scalability and robustness

of FTrack, and the results showed FTrack is promising for real-world deployment.

While FTrack works individually, it can be used to complement any existing indoor localization system to achieve better performance. For our future work, we plan to improve the reliability of user state recognition by introducing more sensor modalities such as gyro and audio.

References

[1] Bruner II G, Kumar A. Attitude toward location-based advertising. *Journal of Interactive Advertising*, 2007, 7(2): 3-15.

[2] Aalto L, Göthlin N, Korhonen J, Ojala T. Bluetooth and WAP push based location-aware mobile advertising system. In *Proc. the 2nd International Conference on Mobile Systems, Applications, and Services*, June 2004, pp.49-58.

[3] Mohan P, Padmanabhan V, Ramjee R. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proc. the 6th ACM Conference on Embedded Network Sensor Systems*, November 2008, pp.323-336.

[4] Thiagarajan A, Ravindranath L, LaCurts K, Madden S, Balakrishnan H, Toledo S, Eriksson J. Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones. In *Proc. the 7th ACM Conference on Embedded Networked Sensor Systems*, November 2009, pp.85-98.

[5] LaMarca A, Chawathe Y, Consolvo S *et al.* Place Lab: Device positioning using radio beacons in the wild. In *Proc. the 3rd International Conference on Pervasive Computing*, May 2005, pp.116-133.

[6] Bahl P, Padmanabhan V. RADAR: An inbuilding RF-based user location and tracking system. In *Proc. the 19th International Conference on Computer Communications*, March 2000, pp.775-784.

[7] Jiang Y, Pan X, Li K, Lv Q, Dick R P, Hannigan M, Shang L. ARIEL: Automatic Wi-Fi based room fingerprinting for indoor localization. In *Proc. the 2012 ACM Conference on Ubiquitous Computing*, September 2012, pp.441-450.

[8] Yang Z, Wu C, Liu Y. Locating in fingerprint space: Wireless indoor localization with little human intervention. In *Proc. the 18th Annual International Conference on Mobile Computing and Networking*, August 2012, pp.269-280.

[9] Constandache I, Bao X, Azizyan M, Choudhury R. Did you see Bob?: Human localization using mobile phones. In *Proc. the 16th Annual International Conference on Mobile Computing and Networking*, September 2010, pp.149-160.

[10] Constandache I, Choudhury R, Rhee I. Towards mobile phone localization without war-driving. In *Proc. the 29th International Conference on Computer Communications*, March 2010, pp.2321-2329.

[11] Ofstad A, Nicholas E, Szcodronski R, Choudhury R. AAMPL: Accelerometer augmented mobile phone localization. In *Proc. the 1st ACM International Workshop on Mobile Entity Localization and Tracking in GPS-Less Environments*, September 2008, pp.13-18.

[12] Jekeli C. Inertial Navigation Systems with Geodetic Applications. Walter de Gruyter, 2001.

[13] Alzantot M, Youssef M. CrowdInside: Automatic construction of indoor floorplans. In *Proc. the 20th International Conference on Advances in Geographic Information Systems*, November 2012, pp.99-108.

[14] Wang H, Sen S, Elgohary A, Farid M, Youssef M, Choudhury R R. No need to war-drive: Unsupervised indoor localization. In *Proc. the 10th International Conference on Mobile Systems, Applications, and Services*, June 2012, pp.197-210.

[15] Xie H, Gu T, Tao X, Lu J. MaLoc: A practical magnetic fingerprinting approach to indoor localization using smartphones. In *Proc. ACM International Conference on Ubiquitous Computing*, September 2014, pp.243-253.

[16] Chung J, Donahoe M, Schmandt C, Kim I J, Razavai P, Wiseman M. Indoor location sensing using geo-magnetism. In *Proc. the 9th International Conference on Mobile Systems, Applications, and Services*, June 28-July 1, 2011, pp.141-154.

[17] Haverinen J, Kemppainen A. Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*, 2009, 57(10): 1028-1035.

[18] Muralidharan K, Khan A J, Misra A, Balan R K, Agarwal S. Barometric phone sensors-more hype than hope! In *Proc. the 15th International Workshop on Mobile Computing Systems and Applications*, February 2014, pp.12:1-12:6.

[19] Wang H, Lenz H, Szabo A, Hanebeck V D, Bamberger J. Fusion of barometric sensors, WLAN signals and building information for 3-D indoor/campus localization. In *Proc. the International Conference on Multisensor Fusion and Integration for Intelligent Systems*, September 2006.

[20] Bendt D J, Clifford J. Using dynamic time warping to find patterns in time series. In *Proc. Advances in Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop*, July 1994, pp.359-370.

[21] Erdös P, Rényi A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 1960, 5: 17-61.

[22] Liu J, Zhong L, Wickramasuriya J, Vasudevan V. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 2009, 5(6): 657-675.

[23] Lord S, Rochester L, Weatherall M, McPherson K, McNaughton H. The effect of environment and task on gait parameters after stroke: A randomized comparison of measurement conditions. *Archives of Physical Medicine and Rehabilitation*, 2006, 87(7): 967-973.

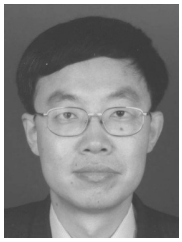
[24] Varshavsky A, LaMarca A, Hightower J, de Lara E. The SkyLoc floor localization system. In *Proc. the 5th IEEE Int. Conf. Pervasive Computing and Communication*, March 2007, pp.125-134.

[25] Montemerlo M, Thrun S, Koller D, Wegbreit B. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. the 18th AAAI National Conference on Artificial Intelligence*, July 28-August 1, 2002, pp.593-598.

Hai-Bo Ye received his M.S. degree in computer science from Nanjing University in 2012. He is currently a Ph.D. candidate in the Department of Computer Science at Nanjing University. His research interests include indoor localization, mobile and pervasive computing.

Tao Gu received his B.S. degree in computer science from Huazhong University of Science and Technology, Wuhan, and M.S. degree in computer science from Nanyang Technological University, Singapore, and Ph.D. degree in computer science from National University of Singapore. He is currently an associate professor in the School of Computer Science and Information Technology at RMIT University. His research interests include mobile and pervasive computing, wireless sensor networks, distributed network systems, sensor data analytics, cyber physical system, Internet of Things, and online social networks. He is a senior member of IEEE and a member of ACM.

Xian-Ping Tao received his M.S. and Ph.D. degrees in computer science from Nanjing University in 1994 and 2001, respectively. He is currently a professor in the Department of Computer Science at Nanjing University. His research interests include software agents, middleware systems, Internetware methodology, and pervasive computing. He is a member of CCF and IEEE.



Jian Lv received his B.S., M.S., and Ph.D. degrees in computer science from Nanjing University in 1982, 1984, and 1988, respectively. He is currently a professor in the Department of Computer Science at Nanjing University. He is also the director of the State Key Laboratory for Novel Software Technology and the vice director of the Institute of Software Technology at Nanjing University. His research interests include programming methodology, pervasive computing, software agent, and middleware. He is a fellow of CCF and a member of ACM.