Wormhole Defense for Cooperative Trajectory Mapping

Wei Chang, Jie Wu, and Chiu C. Tan
Department of Computer and Information Sciences
Temple University, Philadelphia, PA 19122
Email: {wei.chang, jiewu, cctan}@temple.edu

***Abstract*—Cooperative trajectory mapping is an emerging technique that allows users to collaboratively map a person's movements without using GPS measurements in indoor scenarios. This is accomplished by letting users periodically transmit limited amount of information, collected from their mobile phones, to a central server for processing. Cooperative mapping solutions have been proposed for applications ranging from people localization to traffic monitoring. In this paper, we consider the problem of fake trail attacks in cooperative trajectory mapping. In a fake trail attack, the adversary seeks to create a fraudulent shorter path between two locations. The unique characteristics of cooperative mapping make conventional wormhole defenses, such as packet leashes, unsuitable. We propose an efficient algorithm which can successfully deter an adversary from launching such an attack, and we validate the effectiveness of our solution through extensive simulation experiments.**

***Index Terms*—Fake trail, localization, navigation, security.**

I. Introduction

The functionalities of smartphones continue to grow with many such phones now being equipped with hardware such as sensors and accelerometers. Cooperative trajectory mapping is an emerging technique that takes advantage of the capabilities of these smartphones to allow users to combine the data collected from their phones to create maps of user movements in an unknown indoor region. This type of map is known as a *trajectory map*. Trajectory maps are used in various applications such as people localization [1]–[3], public transportation tracking [4]–[6], and traffic monitoring [7].

Cooperative trajectory mapping typically does not use the phone's GPS in order to avoid the weak signal issue in indoor conditions. Instead, the smartphone's accelerometer and compass are used to determine the displacement and moving direction of a user [8]. This data is periodically transmitted to a central depository which collects and processes the readings from multiple users to arrive at a trajectory map. Prior work on cooperative trajectory mapping focuses on building the system and improving map accuracy, and assumes that all participants are honest. However, the presence of malicious users can have a detrimental effect on the final trajectory map.

We can illustrate the effects of a malicious user in Fig. 1. Here, we assume that the trajectory map is used to locate friends in an indoor scenario. In Fig. 1, user A is trying to locate user B. Using the collected data (Figs. 1(a) and (b)), the central depository computes the shortest path from A's current position to B (shown by the arrow lines in Figs. 1(c) and (d); the background map is not available to the server). Assuming that A is walking faster than B, A will eventually catch up with B. In Fig. 1(a), a malicious user, adv, reports a fake path. The depository may use this incorrect information and compute a different path for A. This path is incorrect since it requires A to go though the walls of a room, as shown in Fig. 1(c). Without adequate protection from malicious users, the resulting trajectory map will be inaccurate.

In our paper, we name this type of attack as *fake trail attack* since the adversaries launch the attack by reporting some fake trajectories or fake encounters with other users. The fake trail attack is similar to a wormhole attack, considered in ad hoc networks [9], since we can consider the intersections as nodes and road segments as links of nodes. In a traditional wormhole attack, the adversary will maintain an out-of-band connection between two physical locations in the network. Using this out-of-band connection, the adversary creates a wormhole that can let nodes that are far away from each other appear to be neighbors [10]. As a result, the routing algorithm in ad hoc networks will report an incorrect path between two locations. In our cooperative trajectory mapping problem, an adversary that reports a fake path between two locations will create a wormhole between those two locations. The effect is the same as the traditional wormhole attack since the central depository will use this incorrect information to derive incorrect paths for users.

However, the unique characteristics of cooperative mapping make it difficult to apply existing wormhole defenses.

Firstly, traditional wormhole detection mechanisms assume that every node's sensorial range is R. We can estimate the distance between any nodes based on the neighboring table. In our model, the length of each road segment may be not the same; the number of intersections is not a constant but will grow.

Secondly, conventional wormhole attacks are strongly related to the geometry of the network: most wormhole detection algorithms assume that partial information about a wormhole-free case is available. However, in cooperative trajectory mapping, the original map data is not available to the server.

Lastly, traditional *time and spatial leash*-based wormhole defense approaches let the data packages identify the wormhole; however, in our problem, we have people following the route rather than packets. Unlike packets which can be retransmitted easily, moving people to alternative routes is time

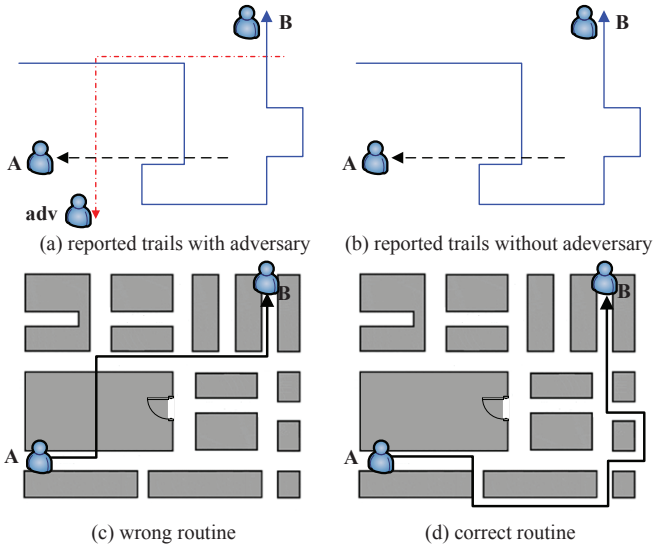


Fig. 1. The effect of a wormhole attack. There are two honest users and one adversary in the example. Fig. 1(a) shows the reported trajectories of the three users. Fig. 2(b) are the reported trajectories without the adversary. The routine results are shown in Figs. 2(c) and 2(d). The shadow areas represent rooms, and we assume that each room has only one door: people cannot go through the rooms.

consuming.

In our paper, a novel centralized algorithm is designed for the detection of fake trail attacks. Considering the uncertainty of a user's instant location, we first estimate the feasible position of every user at time t based on the reported trajectory. After that, we determine whether two users will definitely have physical encounters, may have physical encounters, or will definitely not. Then, we compare our estimation results with the reported encounter records. If there is an inconsistency among the data, we will give some penalty to the users. After observing enough time, we use two thresholds to classify the users into three groups: "honest group", "adversary group" and "suspicious group", and differentially use the data from different groups.

The contributions of the paper are as follows:

1) We are the first to explore the problem of fake trail attacks in the context of cooperative trajectory mapping.
2) We propose a witness-based detection algorithm that is effective against multiple, non-collusive, adversaries. Our algorithm acts as a deterrent to adversaries seeking to attack the system.
3) Extensive simulation experiments were used to validate our solution.

The remainder of the paper is organized as follows. In Section II, we introduce some related work. The system model and attack model are given in Section III. In Section IV, we provide theorems and a corollary for wormhole detection, and a novel wormhole detection and isolation algorithm is presented. The performance analysis and evaluation are described in Section V. We make a conclusion and provide future research in Section VI.

II. Related work

The idea behind the cooperative trajectory mapping problem considered in this paper was proposed by [8]. There, a mobile social network-based navigation system is designed. Each user periodically reports his trajectory and encounter information to the server, then the server can reply with a routine to the user in order to help the user find others. Other work by [11], [12] also used similar ideas for cooperative trajectory mapping. The main difference is that prior work only considers honest users. Our work focuses on *non-colluded*, dishonest users that will intentionally report incorrect paths to disrupt the system.

Detecting fake paths in cooperative trajectory mapping shares similarities with traditional wormhole detection in ad hoc networks. In a wormhole attack, the adversary will attempt to convince other nodes that a path exists between two locations when, in fact, there is no path between the nodes. Using the same methodology as [10], we divide wormhole detection techniques into centralized and decentralized solutions.

Centralized wormhole detections, such as statistic detection [13]–[15] and multi-dimensional scaling [16], [17], rely on the server knowing certain node distributions or network geometric layout information, and using this information to detect the wormhole. However, in our trajectory mapping problem, the server does not have any knowledge about the real map, except for the reported moving trails of users. In fact, the server builds up a map using the mobility paths of users. Thus, existing centralized wormhole detection techniques cannot be applied to our problem.

Decentralized wormhole detection techniques rely on the nodes within the network to monitor the other nodes or data transmissions to detect a wormhole. Work by [18]–[20] requires a node to monitor the topology structure of its neighbors, while [21] lets the node monitor the input and output traffic flows. Other techniques that rely on cryptographic monitoring include packet leashes [22], TESLA [23], and distance bounding [24]. The main difference is that, in our problem, an individual node-reported path may not be used until it can be associated with information from a fixed AP or the global uniform coordinate system. Thus, a node cannot monitor other nodes in our problem.

III. Background

In this section, we will first introduce the system model in our problem. After that, we will discuss how to build a trajectory map based on the system. Finally, the adversary model will be presented.

A. System Model

A cooperative trajectory mapping system has the following three basic components.

1) *A service provider.* The service provider deploys a server to collect user data and to use that information to build and prune the trajectory map. The provider also provides additional services based on the eventual map, such as a friend locator. Except the collected data, the server does not have any domain knowledge about the real map.

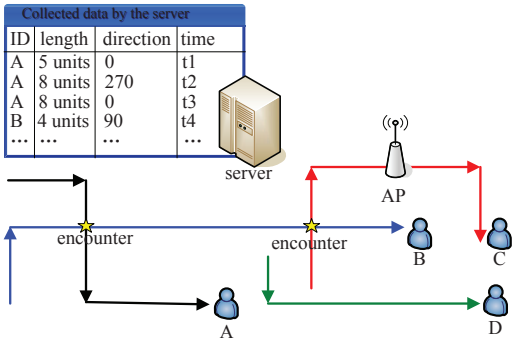


Fig. 2. System model. The system normally contains a central server and several users. Each user periodically reports his trajectory in the form of displacement and direction. The access point is an optimal component, which can increase the chance for error cancelation.

2) *Users.* Users report their trajectories and encounters to the server.
3) *Access point (AP).* An AP is an additional, optional component. It works as a fixed location reference, such as a WiFi access point, whose physical location is known. The AP will periodically broadcast beacons. The purpose of the AP is to quickly establish the spatial relationship between each user's local movement trails and the actual physical coordinates. If all of the reported trails are noise-free and all of the trails are interconnected with each other, the additional AP is useless.

We assume that there are M users that participate in this system and that everyone's clock is loosely synchronized. Each user's mobile phone is equipped with an accelerometer, a compass, a wireless receiver, and an encounter sensor. The accelerometer and compass are used to determine that user's displacement and direction, respectively. The wireless receiver, for instance a WiFi radio, is used to receive beacons transmitted from the AP. The encounter sensor is used to periodically signal the user's presence as well as record the presence of other users. This can be accomplished using a Bluetooth module built into the smartphone. For instance, when user A walks past user B, A's phone will record as "encountering B", while B's phone does the same. We use the UDG (Unit Disk Graph) model to represent the sensorial range of an encounter sensor.

B. Building a trajectory map

A valid user will maintain two lists in his smartphone: *movement list* (ML) and *encounter list* (EL). The ML consists of a serial of displacement from the last recorded position. The tuples in ML are $\{senderID, displacement, direction, time\}$. The encounter information, which is the information about meeting with other users, is obtained by an encounter sensor. When a device learns about an encounter with another device, it will create an entry in the EL, as $\{senderID, time, encounter'sID\}$. The device of each user will periodically report these two lists to the server via a 3G connection.

However, the trajectories of each user are recorded in their own isolated coordinate system, which is only relative to the initial (unknown) location of the user [8]. If all of the trajectories are connected with each other (spatial and temporal encounters), the server can compute the relatively spatial relationships of intersections. Otherwise, we need a public location reference to provide unique spatial information about all of the users such that the spatial encounter near an AP can also be used as a real physical encounter. In a cooperative trajectory mapping system, an AP is randomly set at a place. When a user passes by the communication range of the AP, the user will also record the AP as an encounter. From the trajectories of the users, who have AP encounter records, the users' individual trails can be moved into a uniform coordinate system by joining the places of an AP encounter together. After A has passed by B, who has passed an AP, the trail of A will also join the uniform coordinate system since the spatial relation between A and the AP can be obtained by that of B and the AP, and A and B.

The server applies two variables to store the reported data: the first one is used to store the trajectories which have already joined the uniform coordinate system, and the second one is used for temporarily storing the space-relation unknown trails. Therefore, once a space-relation unknown user can reference to the AP, the server will move his historical trails from the unknown group to the other group.

One application of the trajectory map is a friend locator, which can be accomplished as follows: every user periodically reports his movement traces to the server, and the server stores the reported data. Periodically, the server will use the data to build or update a map by adding the new found paths to the map. Since the server records the current position of each user, the service provider can use the built map and the recorded location to find the shortest routine between users.

C. Adversary Model

The goal of an adversary is to launch a wormhole attack by creating a fake, shorter path between two locations. If the distance between the fake path and the real path is less than a system defined value R_e, then we consider it as an estimation error. We assume that the server has some other algorithm to deal with estimation errors. Thus, the adversary has to select a fake path that deviates from an actual path by at least R_e.

We assume that there are multiple adversaries, but they are all working independently. The number of real users is assumed to be larger than the number of adversaries. An adversary can create a fake path by manipulating the ML or the EL. The adversaries can modify or create any item in their current ML and report them to the server such that some shorter, fake paths will be included in the trajectory map. Adversaries can also modify or create any record in their current EL and report them to the server such that some real users' trajectories will be wrongly added in the map, which may also generate some shorter but nonexistent paths.

In our paper, if the adversary reports a trajectory, which he did not follow, but the trajectory corresponds to an existing

path on the map, we do not regard it as an attack. However, if the associated encounter information with this trajectory is wrong, we view it as a fake trail attack.

IV. Detection algorithm

In this section, we will present our adversary detection algorithm. The structure of this section is as follows: In the beginning, we will present the intuition behind our solution. Then, our adversaries detection and isolation (ADI) algorithm will be discussed in detail. The trajectory-based encounter-type prediction is a key function used in the ADI algorithm. We will also discuss this prediction in this section.

A. Intuition

A key feature of cooperative trajectory mapping is that when a user encounters another user, each user will independently report encountering the other user to the server. The intuition behind our algorithm is to make use of this feature to detect an adversary. We illustrate the intuition using the following example.

Consider an adversary that is physically at location α (loc_α) but reports to the server that he is at loc_β. Since the adversary is not at loc_β, he cannot determine whether there are any users at loc_β. Considering that there is a large number of users in real applications, the probability of guessing correctly is very small, and we do not consider this case. Now, let us assume that an honest user is at loc_β. Since the adversary is not at loc_β, the honest user does not report encountering the adversary. This results in an inconsistency, or an *abnormal event*, since a supposed encounter did not occur. The honest user hence acts like a *witness* that can be used to identify an adversary. Over time, an adversary will be associated with more abnormal events than honest users, and the server will use this as a means of identifying adversaries.

Based on the above idea, in this paper, we present the adversaries detection and isolation (ADI) algorithm. In the ADI algorithm, we assign a suspected degree to each user. The suspected degree presents the percentage of abnormal events to the number of encounters. The algorithm periodically checks the intersections of each pair of users, estimating the possible *encounter-type* of the users. There are three types of encounters: two users cannot encounter, may encounter or must encounter with each other. Then, we use the estimated encounter-type to compare with the reported encounter information. By this way, we can find the abnormal cases: (i) must meet with no encounter, (ii) impossible meet with an encounter(s), and (iii) those with a single-sided encounter record. We update the suspected degree based on the detected abnormal events. After enough data is collected, we use pre-defined thresholds to find out the identities of the adversaries, suspicious and honest users.

The key challenges behind our algorithm are: (a) how to correctly determine abnormal events from a set of trajectories and encounters. Since the exact movement patterns of users are unknown and the server is only given the displacements and its corresponding time, there is uncertainty about the happening of encounters; (b) how to correctly assign a suspicious degree to the found abnormal cases since assigning wrong suspicious degree may cause an honest user to be regarded as an adversary.

B. Adversaries Detection and Isolation (ADI) Algorithm

When the server runs the ADI, it traverses all displacements and encounter records to find whether there are abnormal cases in which the trajectory information is inconsistent with the encounter records, as shown in Algorithm 1.

Algorithm 1 Adversaries detection and isolation algorithm (ADI)

```
SD.suspect= 0, SD.witness= a small non-zero number
for each time period T do
    if the result of TBEP (Algorithm 2) or encounter records satisfy the requirement of being a witness then
        SD.witness=SD.witness+1
    if the result of TBEP conflicts with encounter records then
        SD.suspect=SD.suspect+1
Find the suspects by threshold
Build trajectory map without using the data of suspects
Return: identities of suspects and the cooperative trajectories map
```

In the beginning of the ADI algorithm, each pair of users u and v is assigned with a suspected degree $SD(u, v)$.

$$SD(u, v) = \frac{SD.suspect}{SD.witness}$$

The initial value of $SD(u, v)$ is zero. The suspected degree consists of two parts. The first part (SD.suspect), the numerator, is a suspected penalty counter. The ADI algorithm will check every users' reported displacements. Based on the displacements, we can estimate the possible encounter-type of the users. The trajectory-based encounter prediction algorithm (TBEP) will be presented in the later part of our paper. Once the algorithm detects an inconsistency between the encounter prediction result and the reported encounter information, the $SD(u, v)$ of the users will be given a penalty. Admittedly, an adversary with knowledge of the ADI algorithm might intentionally frame honest users by reporting false encounters. We will discuss some details in Section VII. The second part (SD.witness), the denominator of the suspected degree, records the times of being a witness: (i) the user's trajectory encounter with others' trajectories, or (ii) there is an encounter report about the user.

After observing the reported information for a long enough time, we can classify the users into three groups: an "honest group", a "suspicious group", and an "adversary group" by using two pre-defined cluster thresholds, threshold 1 and threshold 2 (threshold 1 $>$ threshold 2). If the value of $\sum_v SD(u, v)$ is above threshold 1 or below threshold 2, the corresponding users belong to the "adversary group" or the "honest group", respectively. The remaining users fall into the "suspicious group", which needs to be further investigated (observing more time). In the phases of building the cooperative trajectory map, we first build the map only using

the trajectories of the honest users. Then, we only partially and temporarily revise the map if there is no path between two users without the corresponding paths from the suspicious group' trails. We do not use the data from the adversary group.

C. Trajectory-based Encounter-type Prediction (TBEP)

The function of the TBEP algorithm is to estimate the encounter-type of any two given trajectories. In this part, we will first present the definition of some terms which we will use. Then, we will introduce the TBEP algorithm. Finally, some related theorems will be presented. The details of the help functions, which the TBEP algorithm calls, will be discussed in the next section.

We will illustrate TBEP by using the example in Fig. 3. There are two displacements of users A and B (Fig. 3(a)). We assume that they finished the displacements at the same time. If we can obtain the instant speed of the users at any time, we can clearly find out when and where those two users met with each other in the past. However, determining the exact position of each user at any point in time is difficult to obtain without knowing the instant speed. Knowing the instant speed of a user is difficult to obtain in a realistic scenario.

However, considering the maximum speed limitation, we still can find a user's feasible position bound at any time. The *feasible position bound* is a time related function, which restricts a user's possible position at a given time t by indicating the farthest and nearest locations of where the user could have reached. If the user moves along a line, there are two end points of the position bound. The first point is used to describe that the user waits at the beginning, then moves to the destination with maximum speed; the second point is used to describe that the user first moves with maximum speed, then waits at the destination. As a result, the feasible position of a user at time t will be a segment.

***Definition** 1: for any displacement, going along the moving direction, we name the farthest position that the user can get to at time t as the head end point, and the corresponding sensor area as head sensor area; we name the nearest position as the tail end point and call its sensor area as the tail sensor area.*

Fig. 3(b) illustrates the feasible position bounds of one user. The user's real moving pattern could be any lines inside the parallelogram, except that the slope of the line must be smaller or equal to the slope of bounds. Since each user is equipped with an encounter sensor, the shape of the possible sensor region of a user at any time is similar to an *ellipse*, which is the union of a group of circles whose center is located at the possible position segment, as shown in Fig. 3(c).

TBEP algorithm. As one of the key building blocks of our solution, the TBEP algorithm presents the way for the server to determine the encounter-type of the given trajectories, which can be further used to determine whether abnormal events occur. After receiving the respective displacement of two users, the server first uses the position function ($\vec{p}()$) to estimate the latest moving time and the fastest arriving time of a user. Then, the server applies the Possible Trajectory Boundary function($PTB()$) to compute the users' feasible position bounds in X-Y-T space as: the space function of A's head end point, $SF_H(A)$; the space function of A's tail end point, $SF_T(A)$; the space function of B's head end point, $SF_H(B)$; the space function of B's tail end point, $SF_T(B)$.

TABLE I
TABLE OF NOTATION FOR TBEP

t_B	beginning shared time of two displacements
t_E	end shared time of two displacements
$\{(x_i^A, y_i^A, t_i^A)\}$	a series of displacements of user A in T
S_{max}	maximum speed
EC	potential encounter conditions and time
d	length of a displacement
SF_H	head boundary function
SF_T	tail boundary function
ER	encounter records
R	sensing range
$PTB()$	possible trajectory boundary function
$IT()$	time-space relation function

Algorithm 2 Trajectory-based encounter prediction (TBEP) algorithm

```
Input:{(x_i^A, y_i^A, t_i^A)}, {(x_i^B, y_i^B, t_i^B)}, S_max, R
Output: EC
Compute SF_H and SF_T of both A and B, by run PTB({(x_i^A, y_i^A, t_i^A)}, {(x_i^B, y_i^B, t_i^B)}, S_max, R).
if IT({(x_i^A, y_i^A, t_i^A)}, {(x_i^B, y_i^B, t_i^B)}, R) can encounter in corresponding time then
    EC.Type ="must meet", EC.Time = corresponding time
else
    if IT({(x_i^A, y_i^A, t_i^A)}, {(x_i^B, y_i^B, t_i^B)}, R) cannot have encounter in corresponding time then
        EC.Type ="cannot meet", EC.Time = corresponding time
    else
        EC.Type ="may meet", EC.Time = corresponding time
Sort ER's tuples according to encounter type and time.
Return: encounter conditions
```

***Theorem** 1: If all of the distances of $\{SF_H(A), SF_H(B)\}$, $\{SF_T(A), SF_H(B)\}$, $\{SF_H(A), SF_T(B)\}$ and $\{SF_T(A), SF_T(B)\}$) are greater than R in this time period, the two users cannot meet with each other, and therefore the server should not contain the corresponding encounter records, as shown in Fig. 4(a)*

Proof: It is obvious since all of the distance between the two users are greater than the R. ■

***Theorem** 2: If some of the distances, but not all of the distances of $\{SF_H(A), SF_H(B)\}$, $\{SF_T(A), SF_H(B)\}$, $\{SF_H(A), SF_T(B)\}$ and $\{SF_T(A), SF_T(B)\}$) are greater than R in this time period, the two users may meet with each other.*

Proof: It is also obvious since some trajectories of the two users can encounter with each other and some cannot. ■

***Theorem** 3: If all of the distances of $\{SF_H(A), SF_H(B)\}$, $\{SF_T(A), SF_H(B)\}$, $\{SF_H(A), SF_T(B)\}$ and $\{SF_T(A), SF_T(B)\}$) are less than R, or are equal to R in this time period, the two users cannot meet with each other, and therefore the server should contain the corresponding encounter*

records.

Proof: The feasible position bounds of the two users are two parallelograms in X-Y-T space. We name the feasible position bound in X-Y-T space as the *Feasible Position Region*. Since the space is continuous and all of the edges satisfy the conditions for encountering with each other, any pair of lines inside the parallelograms must encounter with each other. ■

The relation between R and the distance of two lines can be computed by the Intersection function($IT()$).

V. Technical Details

In this section, we will present the technical details about our solution. First, we will introduce the position function which we used to indicate the feasible position of a user at a given time instance in X-Y coordinates. The possible trajectory boundary function is the extension of the position function in X-Y-T space, which will be discussed at the second part of this section. After that, the intersection function will be presented. We use the intersection function to estimate the encounter conditions of two users' feasible positions. In the last part of the section, we will define what is an abnormal event and will explain why we classified the entire encounter conditions into three groups: "probably honest", "probably dishonest", and "uncertain".

A. Position function ($\vec{p}()$)

Given one displacement of a user's trajectory, the location of the user can be presented by $\vec{p}(t)$, as shown in Fig. 3(b). Suppose that a user reported one displacement from location $\vec{p}(t_1)$ to $\vec{p}(t_2)$ at time from t_1 to t_2 ($t_1 < t_2$), and the length of this displacement equals d. We also assume that the speed range of a user is $[0, \vec{S}_{max}]$.

$$\vec{p}(t) = \begin{cases} [\vec{p}(t_1), \vec{p}(t_1) + \vec{S}_{max}(t - t_1)], \\ \quad t \in \left[t_1, t_2 - \frac{d}{|\vec{S}_{max}|}\right] \\ [\vec{p}(t_1) + \vec{S}_{max}(t - t_2 + \frac{d}{|\vec{S}_{max}|}), \vec{p}(t_1) + \vec{S}_{max}(t - t_1)], \\ \quad t \in \left(t_2 - \frac{d}{|\vec{S}_{max}|}, t_1 + \frac{d}{|\vec{S}_{max}|}\right] \\ [\vec{p}(t_1) + \vec{S}_{max}(t - t_2 + \frac{d}{|\vec{S}_{max}|}), \vec{p}(t_2)], \\ \quad t \in \left(t_1 + \frac{d}{|\vec{S}_{max}|}, t_2\right] \end{cases}.$$

In order to simplify the solution, we add a new dimension "time" into the traditional X-Y coordinates. Therefore, Fig. 3(a) can be converted to Fig. 4. The displacement of each user is represented by a parallelogram in X-Y-T space. The projection of the parallelogram on the X-Y plane is the same segment in Fig. 3(a). The lines, which are parallel with the t-axis, represent the situation of not moving. Because of the speed limitation, the slope between any two points in the same parallelogram should be less or equal to the maximum speed. The real trajectory of the user in X-Y-T space will be a curve in the parallelogram. Hence, if A and B meet with each other, their real trajectories must intersect in X-Y-T space. In X-Y-T space, the position function $\vec{p}(t)$ will be converted to the *possible trajectory boundary function ($PTB()$)* .

TABLE II
Table of notation for helper functions

$\vec{p}(t)$	user's position at t in system's coordinates
S_{max}	the maximum speed
(x_i^A, y_i^A, t_i^A)	a piece of displacements of user A in T
$(x_1^A(t), y_1^A(t))$	location where A started the displacement
$(x_2^A(t), y_2^A(t))$	location where A ended the displacement
t_1^A	time that user A began the displacement
t_2^A	time that user A finished the displacement
d	length of a displacement
$(x^A(t), y^A(t))$	possible location of A at given time t
(x^A, y^A)	the location of A at any time
R	sensing range
a,b,c,d	temporary parameters

B. Possible trajectory boundary (PTB()) function

The PTB() function provides the two position bounds of a user in X-Y-T space. It has four inputs: $\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}, S_{max}$ and R. The user first moves with maximum speed to finish the reported displacement, then stays still at the end of the displacement; the user can also wait first at the beginning point, then finish the displacement just on time. Therefore, there are two trajectory boundary functions. The head boundary function is:

$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - t_1^A}{d/S_{max}}$$

The tail boundary function is:

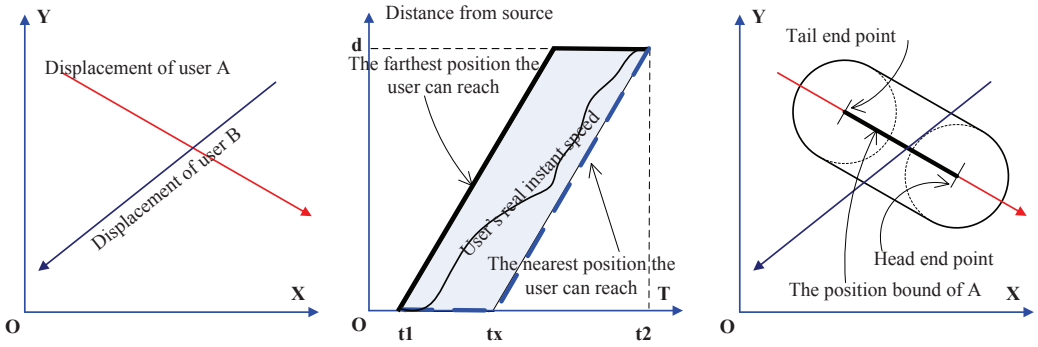
$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - (t_2^A - d/S_{max})}{d/S_{max}}$$

Given a pair of displacements, if we know that the users covered the displacements by a constant speed, we can use the *intersection function (IT())* to check whether an encounter happened.

C. Intersection (IT()) function

We use the IT() function to determine whether the two given users can encounter each other. It has three inputs: $\{(x_i^A, y_i^A, t_i^A)\}, \{(x_i^B, y_i^B, t_i^B)\}$ and R. Assume that there are two users named A and B. They all report one displacement, and we also assume that they all moved at maximum speed S_{max}. We assume that the beginning time is t_1^A, and the end time is t_2^A. At the same time, the initial position is (x_1^A, y_1^A), and the end of the displacement is (x_2^A, y_2^A). Therefore, the displacement of A can be represented as $((x_1^A, y_1^A, t_1^A), (x_2^A, y_2^A, t_2^A))$. Similarly, we also have B's displacement as $((x_1^B, y_1^B, t_1^B), (x_2^B, y_2^B, t_2^B))$. Now we want to compute when and where these two users can generate the encounter records. Firstly, the trajectory of a user can be represented as follows in a 3-D space, where the x-axis is one possible moving direction and the y-axis is the perpendicular direction to the x-axis. The third direction is time.

$$\frac{x^A - x_1^A}{x_2^A - x_1^A} = \frac{y^A - y_1^A}{y_2^A - y_1^A} = \frac{t^A - t_1^A}{t_2^A - t_1^A}$$



(a) two reported trajectories (b) the position bound of a user (c) the possible sensor region of A

Fig. 3. Relation between two users' trajectories. Fig. 3(a) two trajectories with spatial intersection. Fig. 3(b) user's feasible position bound; the user's real instant speed could be any line inside the shadowed parallelogram, whose slope is smaller or equal to the maximum speed. Fig. 3(c) user's possible sensor range at time t.

The trajectory of B from t_1^B to t_2^B can be represented as follows:

$$\frac{x^B - x_1^B}{x_2^B - x_1^B} = \frac{y^B - y_1^B}{y_2^B - y_1^B} = \frac{t^B - t_1^B}{t_2^B - t_1^B}$$

Hence, to a given time t, whether a piece of encounter record will be generated or not can be determined by the truth value of following formula:

$$(x^A(t) - x^B(t))^2 + (y^A(t) - y^B(t))^2 \leq R^2$$

After using t to represent $x^A(t)$, $x^B(t)$, $y^A(t)$ and $y^B(t)$, we can get:

$$(a^2 + c^2)t^2 + 2(ab + cd)t + (b^2 + d^2 - R^2) \leq 0,$$

where:

$$a = \frac{x_2^A - x_1^A}{t_2^A - t_1^A} - \frac{x_2^B - x_1^B}{t_2^B - t_1^B}$$

$$b = -(\frac{x_2^A - x_1^A}{t_2^A - t_1^A})t_1^A + (\frac{x_2^B - x_1^B}{t_2^B - t_1^B})t_1^B + x_1^A - x_1^B$$

$$c = \frac{y_2^A - y_1^A}{t_2^A - t_1^A} - \frac{y_2^B - y_1^B}{t_2^B - t_1^B}$$

$$d = -(\frac{y_2^A - y_1^A}{t_2^A - t_1^A})t_1^A + (\frac{y_2^B - y_1^B}{t_2^B - t_1^B})t_1^B + y_1^A - y_1^B$$

D. Abnormal Events

Here, we define what is an abnormal event. From the view of the server, it can observe two things: trajectories and encounter records. The trajectories of two users have three encounter-types: must meet, potentially meet, and cannot meet. The encounter records about two users at a given time can also be classified into three kinds: no encounter, single encounter, and two encounters. Based on the given trajectories, we can classify the observations made by the server into three groups: an "uncertain" group, a "probably honest" group, and a "probably dishonest" group for all of the 9 kinds of combinations. An abnormal case will fall under the "probably dishonest" group.

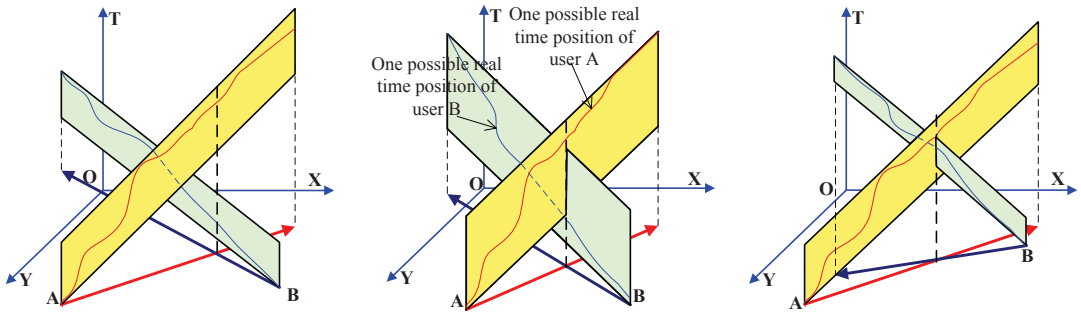
The "probably honest" group represents the conditions: (i) at a given time t, the position bounds of two users satisfy the condition of "must meet," and there is a pair of accordant encounter records in the server; or, (ii) the trajectories of two users satisfy the case of "may meet", and a pair of consonant encounter records is stored in the server. If both of the two users are honest, the happening of the conditions (i) and (ii) is obvious. However, if one of the users or both of them are the adversaries, then they can only be classified as "probably honest" only if the adversary can correctly guess the identity of the other party. The probability of this is negligible.

The "uncertain" group also consists of two conditions: (i) the trajectories of two users satisfy the case of "may meet", and there is not any encounter record; or, (ii) the trajectories indicate the condition of "cannot meet", and there is no encounter record. We cannot conclude anything about the identities from these two cases. If both of them are honest users, these two conditions can happen; If there is one or two adversaries, the two conditions can still happen with a high probability. For the condition (i), when there is only one adversary, the adversary can report a fake path, which has a spacial intersection with the path of the honest user, and the real path of the adversary does not encounter the honest user's path. If both of them are the adversaries, they just report two spacial intersected paths, but they do not encounter each other in fact; then, the condition (i) happens. As for the condition (ii), if there is a pair of "impossible meet" paths and the real path of one adversary does not encounter the other user, no matter whether the identity of the other one is honest or not, condition (ii) can happen.

Except the above 4 conditions, the remaining 5 conditions belong to the "probably dishonest" group where at least one of the users could be an adversary.

***Theorem** 4: an adversary is more likely to be associated with an abnormal case than an honest user if the number of real users is greater than that of the adversaries.*

Proof: assume that p is the probability that two users meet with each other at a given time. Suppose that the number of



(a) cannot encounter with each other (b) may encounter with each other (c) must encounter with each other

Fig. 4. Illustration of encounter-type in X-Y-T space. The parallelograms represent the feasible position regions of users. Any line inside the region, whose slop is small or equal to the maximum speed, may be the real instant position of the user. Fig. 4(a) two trajectories cannot have encounter records. Fig. 4(b) two trajectories may have encounter records. Fig. 4(c) two trajectories must have encounter records.

real users is a, the number of the fake users is b, and $a >> b$. For a real user, the reason for having abnormal data is that he encountered the adversaries' real paths or fake paths. Hence, given the condition that a real user meets with another user, the conditional probability that the given user meets an abnormal event is $2b/(a-1+2b)$. However, given an adversary, if his real trajectory comes across the trajectories of real users or other adversaries' real trajectories, or if the given adversary's fake trajectory meets with the trajectories of real users or other adversaries' fake trajectories, he may encounter inconsistent conditions. Therefore, given the condition that a fake user meets with another user, the conditional probability that the given user potentially meets an abnormal event is 1. ■

VI. Performance Analysis and Evaluation

In this section, we will first describe our simulation setup. Then, our evaluation metrics will be presented. The simulation results and analysis will be given at the last part of the section.

A. Experimental Setup

We use Matlab to make the simulation. The users' trajectories are generated by Levy walks mobility model [25] and users' real traces [1]

In the simulation based on synthetic data, we first randomly generate 50 test groups, and each group contains the trajectories of 50 users. We let the period of the location updates be 60 sec, and we observe each user for $6,000$ time units. We set the size of the target area as 500×500 distance units. The silence time period is changed from 0 to 10 seconds. The mobility model's parameter α equals 1, and β also equals 1. We also set the minimum flight length as 60 distance units, and we set the maximum flight length as 100 distance units. When users get the boundary of the target region, we use *reflection* to ensure that the total number of users do not change.

We assume that the encounter sensor range is R. In the simulation, we mainly use $R = 5$ distance units. The maximum speed of users is represented by S_{max}, and we mainly let it be 3 distance units per second. In each time unit, the user randomly picks a speed from 0 to S_{max} to be the real moving speed in the time unit. During simulation, we randomly select N_{Real} users' traces as the real users' trajectories. We select N_{AR} users' traces as the fake trajectories of adversaries, and we also select N_{AF} users' trajectories as the real hidden trajectories of the adversaries $N_{AR} = N_{AF}$. After dividing the trajectory data set, we first compute the encounter records if adversaries report their real traces, which also means there are no adversaries.

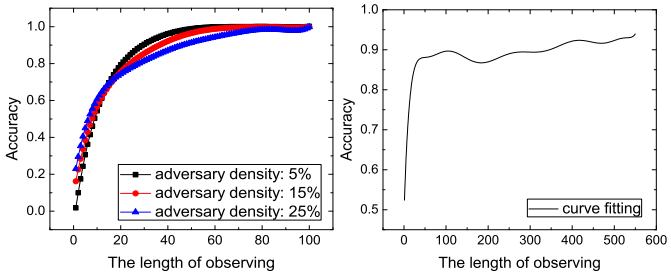
During the computation of encounter records, we assume that each user finishes the displacements by the average speed in this segment. When the distance between two users is less or equal to the sensorial range R, we record the identities of these two users, and we also record the beginning and the end time of this encounter record. Then, we observe the displacements of users for a period of time. In each time period, we use the reported trajectories of users to check whether there are some abnormal cases. If we find an abnormal case, we will give a penalty to both of the users. After observing the traces for a pre-defined period of time, we rank the suspecting list in descending order and pick the first $N_A R$ users as the adversaries.

In the simulation based on real data, we have the moving trajectories of 35 users. We randomly set the identity of users as real users' trajectories, adversaries' real trajectories, and adversaries' fake trajectories. For example, if there are 5 adversaries, then we randomly choose 5 from the data set as their real moving paths, and we randomly choose another 5 as their reported fake trails. The data set, which we used, records the positions of a user in $2,604$ reporting time units. The setting for the rest of the parameters is the same as the setting in the synthetic data based simulation.

Considering the length of observation, the adversaries' density, the users' density, and the length of users' sensorial ranges will effect the accuracy of the adversary detection algorithm; we use the simulations to analyze the following relations:

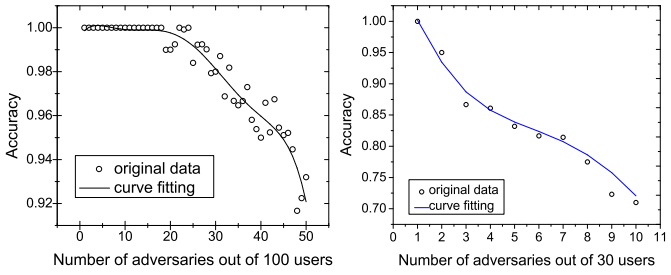
1) The relationship between the length of observation time

[1]The real data comes from RAWDAD, which is a community resource for archiving wireless data at Dartmouth. The data set we used names ncsu/mobilitymodels/GPS20090723922009072320 09-07-21. The data was collected by GPS from NCSU, North Carolina, USA, 2009.



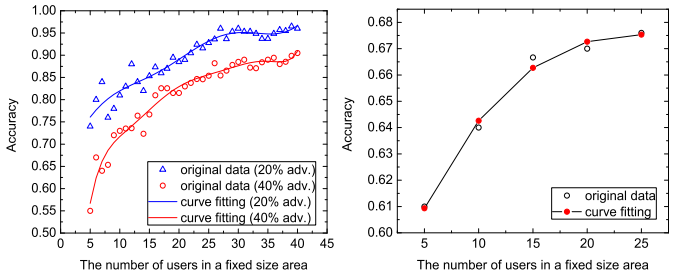
(a) synthetic data (b) real data

Fig. 5. Observing time VS accuracy.



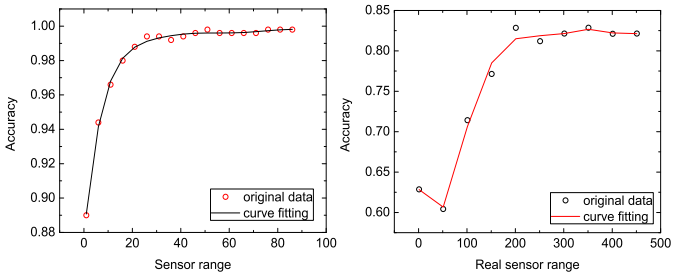
(a) synthetic data (b) real data

Fig. 6. Adversary density VS accuracy.



(a) synthetic data (b) real data

Fig. 7. User density VS accuracy.



(a) synthetic data (b) real data

Fig. 8. Sensor range VS accuracy (beginning part).

and the accuracy of our proposed detection algorithm.

2) The relationship between adversaries' density and the accuracy of our proposed detection algorithm.
3) The relationship between users' density and the accuracy of our proposed detection algorithm.
4) The relationship between the length of users' sensorial ranges and the accuracy of our proposed detection algorithm.

B. Evaluation Metric

Our evaluation was independently made in two different assumptions, and therefore two metrics were used in our evaluation. In the first part of our simulation, we assume that the central server somehow knows the total number of adversaries. Although this assumption seems unrealistic, our simulation results are still valuable if the social scientists can predict the average number of adversaries by some statistic model since the auxiliary information will increase the speed and the accuracy of detecting the adversaries. The first metric we used to evaluate our algorithm is termed *Accuracy*. This is computed as the following:

$$Accuracy = \frac{\text{Number of successfully detected}}{\text{Total number of adversaries}}$$

The second part of our evaluation is based on the assumption that there is no auxiliary information about the number of adversaries. During the simulation, we use the cluster algorithm to automatically classify the users into two groups based on their suspected degree, which is computed by our algorithm. The metrics we applied in this part are False Positive Rate (FPR) and False Negative Rate (FNR).

$$FPR = \frac{\text{FP}}{\text{FP+TN}}, \quad FNR = \frac{\text{FN}}{\text{TP+FN}}$$

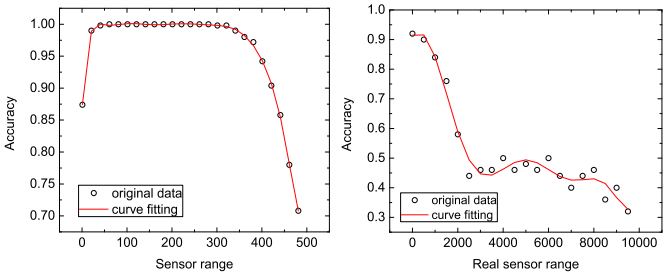
where FP is the number of false positive, FN is that of false negative, TP represents the total amount of true positive, and TN represents that of true negative.

C. Simulation Results

Fig. 5 indicates the relationship between the accuracy of our proposed algorithm and the length of observing time. In the synthetic data-based simulation, which is shown in Fig. 5(a), we randomly select 40 users from each test group as the real ones, and we also randomly pick 5%, 15%, and 25% of users as the adversaries. We use our algorithm to detect the wormhole makers by observing the traces of users for a period of time (100×60 seconds). We repeat our experiments 50 times and use the average value of results to draw Fig. 5. From the simulation results, we could find that the accuracy of the proposed algorithm is increased with the growth of observing time. From an analytical point of view, as the observing time increases, there will be more intersections between users' trajectories in temporal-space. Hence, we can obtain more "witness points", which may contain the abnormal cases.

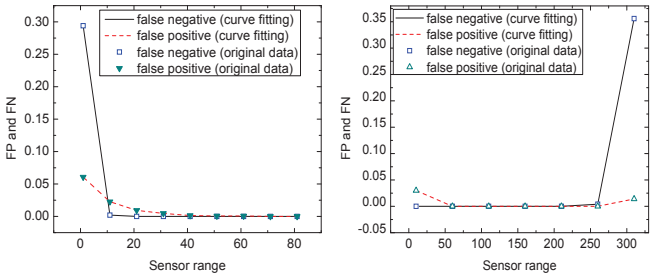
Fig. 5(b) shows the relationship between the length of observing and the accuracy in the real trace. In our experiment, we randomly select 25 users as the real ones and 5 users as the adversaries. We set the sensor range as 350 distance units. We repeat our experiment 50 times and get Fig. 5(b) as our result. This result is consistent with the synthetic data's result.

However, having more observing time also costs more resources to compute. In order to compare the effectiveness of other components, in the later simulation, we always use 50×60 seconds as the default length of observing time, and we use 350 time units as the default length of observing time in real trace-based experiments.



(a) synthetic data (b) real data

Fig. 9. Sensor range VS accuracy (overall).



(a) beginning part (b) overall

Fig. 11. Sensor range VS false positive and false negative.

The ratio of adversaries out of the total number of participants can be used to evaluate the influence on accuracy. Fig. 6(a) represents the relation between this ratio and the detection accuracy by using synthetic traces. With the increasing number of adversaries, the accuracy of detection goes down since the trust of users becomes less and less. Fig. 6(b) is our experiment based on real data. We repeat our simulation 30 times.

The density of users in a region is also an important factor which influences the accuracy of detection. Fig. 7(a) illustrates the relation between participant density and the accuracy. In the simulation, we randomly generate 50 test data sets, and we respectively check the accuracy from density equaling 5 per 500×500 area to 40 per 500×500 area. Also, we assume that the percentage of adversaries is 20% and 40%. Fig. 7 is the average result of our simulation. From the image, we can realize that a higher density of users will bring more accuracy, which can be verified by our experiment results based on real traces, which is shown in Fig. 7(b).

Figs. 8 and 9 show the relationship between the encounter sensor range and the detection accuracy. In the synthetic data-based simulation, we select 30 users' trajectories as the traces of real users in each test group, and we select another 10 users' trajectories as the real traces of the adversaries. Then, we obverse 60×60 seconds. Since our proposed wormhole detection and isolation algorithm is based on abnormal cases in the encounter area, the accuracy of our algorithm will be increased when there are more encounter conditions. Using a larger sensorial range R will provide more chances for encountering the other users. However, if the value of R exceeds a value, the accuracy will become less since there may always be abnormal events in the region. In the simulation, we first let R change from 1 distance unit to 500 distance units. From Fig. 9(a), we can clearly see that the accuracy is increased with the growth of the size of the sensor region in the beginning part. Then, the accuracy maintains stabilization, and it goes down after R is larger than 320. In order to clearly observe the growing pattern in the beginning part, we make another simulation. This time, we let R vary from 1 to 90 distance units, and the simulation result is shown in Fig. 8(a). The results of the real traces are respectively shown in Figs. 8(b) and 9(b). Each data point in the picture is the average value of 50 test results.

In the second part of our simulation, we assume that the total number of adversaries is not available to the center server. After observing the collected data for a while, the server uses the "max cluster" algorithm on the suspected degree data set. In order to increase the difference between honest users and adversaries' suspected degrees, we increase the length of our simulation's time to at least 300 time units (300×60 seconds). We repeat each simulation 50 times.

Fig. 10 shows the relationship between the length of observation time and false positive, false negative. All of the simulation settings are the same as the settings in Fig. 5, except that the observing time is 500 time units. From the simulation results, we can clearly see that observing longer will increase the accuracy of detecting the adversaries, and high adversary density requires more observing time to get an accurate result.

Fig. 11 indicates the relation between the encounter sensor range and the accuracy in false positive and false negative formations. Similar to previous evaluations, we made two simulations. We first checked the change pattern in the accuracy from 1 to 81 distance units, then we tested the overall accuracy changing pattern in which the sensor range varied from 10 to 310. The simulation results are consistent with our first part's results.

Fig. 12 illustrates the relationship between the accuracy and the adversaries' density. In the simulation, we changed the adversary density from 1% to 25%. Our simulation shows that our algorithm has very high accuracy when the adversary density is less than 15%. Higher adversary density requires the server to observe the data for a longer period of time.

Our last simulation is about the user density in an area of fixed size. Although the adversary density may be the same, a higher user density can cause more chances for having encounters with others. In the simulation, we set the adversary

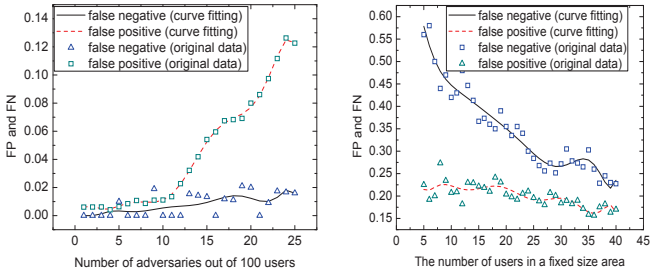
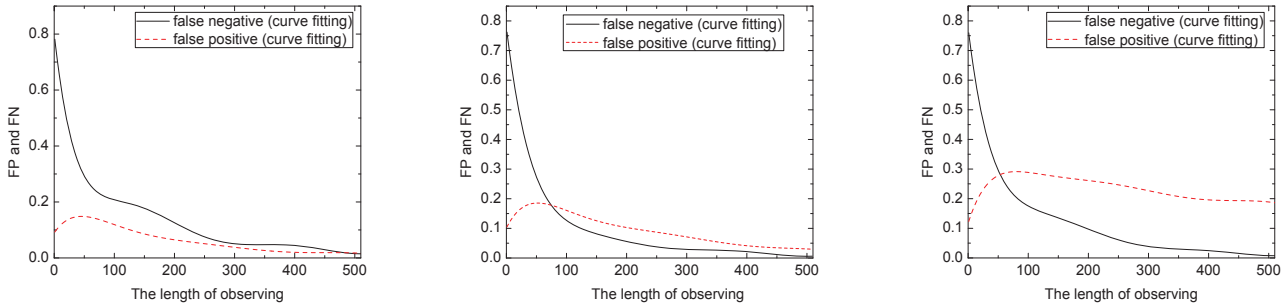


Fig. 12. Adv. density VS FP & FN. Fig. 13. User. density VS FP & FN.



(a) 5% adversary density (b) 15% adversary density (c) 25% adversary density

Fig. 10. Observing time VS false positive and false negative.

density as 20%. Our simulation results are in Fig. 15, which shows that a higher user density can increase the speed of adversary detection, and it can also increase the accuracy of the detection.

In summary, we considered four key factors which can influence the accuracy of our algorithm. We found that observing the trajectories longer will improve accuracy since this increases the probability of abnormal events. If the total number of users is fixed, then increasing the number of adversaries in a target region reduces the accuracy of our algorithm. When the user density becomes larger, the accuracy of detection will also be increased since it increases the probability of having encounters in a fixed size region. We also determine that increasing the users sensing range will also improve the accuracy of our detection algorithm.

VII. Further discussion

Here, we discuss some additional issues regarding our solution:

Adversary reporting behavior. In our paper, we consider that the adversaries continuously report some fake locations. An alternative is for the adversary to mix some real locations with fake locations to try to avoid detection. However, the server can easily detect the abnormal data: the distance between two reported locations cannot be larger than the maximum speed. If the adversaries only report some fake locations, which are close to the real ones, the fake paths may not result in a shorter, nonexistent path. Hence, we only consider the case that the adversaries continuously use fake trajectories.

Effect of the number of users on detection. Our method does not require a minimum number of users in a fixed size area in order to detect dishonest users, but we do require that the number of honest users must be larger than that of the dishonests. From our simulation part, we have already seen that the smaller the adversary density is, the better our method's performance is.

Adversary framing the honest user. Instead of creating a fake and short path, the adversary can also try to frame the honest users by maliciously reporting a false encounter. Our method can deter this condition since our inconsistency penalty counts on both users. Framing other users will also expose the adversary himself.

Adversary's long, fake path. One attack that is difficult to defend against is when the adversary reported a nonexistent, relatively longer path between two locations. Later, as new trajectories are added to the server, that nonexistent trajectory may become part of the shortest path between the two locations; the case rarely happens if the user density is large enough: the probability of encountering with nobody is very small. In order to solve the case, during the phase of building the trajectory map, we can only use the road segments which have already been covered by many users. By this way, the unconcluded adversary cannot launch the attack.

VIII. Conclusion

In this paper, we consider the problem of wormhole attacks in cooperative trajectory mapping. We propose a witness-based detection and isolation algorithm that is effective against multiple, non-collusive, adversaries. We use extensive simulations to validate our solution. Our future work intends to consider two extensions. The first is that we intend to explore a stronger attack where the adversaries can collude. One possibility is to combine secure localization techniques that change the functionalities of the APs together with our witness-based approach to detect colluding adversaries. The second extension is to build a real system and evaluate our solution.

Acknowledgment

This research was supported in part by NSF grants ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, and CCF 0830289.

References

[1] I. Constandache, R. Choudhury, and I. Rhee, "Towards mobile phone localization without war-driving," in *IEEE INFOCOM*, 2010.
[2] P. Bahl and V. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *IEEE INFOCOM*, 2000.
[3] Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm, "Accuracy characterization for metropolitan-scale Wi-Fi localization," in *ACM MobiSys*, 2005.
[4] A. Repenning and A. Ioannidou, "Mobility agents: guiding and tracking public transportation users," in *ACM AVI*, 2006.
[5] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. Landay, "UbiGreen: investigating a mobile tool for tracking and supporting green transportation habits," in *ACM CHI*, 2009.
[6] S. Carmien, M. Dawe, G. Fischer, A. Gorman, A. Kintsch, J. Sullivan, and F. James, "Socio-technical environments supporting people with cognitive disabilities using public transportation," *ACM TOCHI*, 2005.

[7] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *ACM SenSys*, 2008.

[8] I. Constandache, X. Bao, M. Azizyan, and R. Choudhury, "Did you see Bob?: human localization using mobile phones," in *ACM MobiCom*, 2010.

[9] Wu, J., *Handbook on theoretical and algorithmic aspect of sensor, ad hoc wireless, and peer-to-peer networks*. Auerbach Publications, 2006.

[10] L. Buttyán and J. Hubaux, *Security and cooperation in wireless networks*. Cambridge University Press, 2007.

[11] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "VTrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *ACM SenSys*, 2009.

[12] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *ACM SenSys*, 2010.

[13] L. Buttyan, L. Dora, and I. Vajda, "Statistical wormhole detection in sensor networks," *Security and Privacy in Ad-hoc and Sensor Networks*, 2005.

[14] N. Song, L. Qian, and X. Li, "Wormhole attacks detection in wireless ad hoc networks: A statistical analysis approach," in *IEEE IPDPS*, 2005.

[15] L. Qian, N. Song, and X. Li, "Detection of wormhole attacks in multipath routed wireless ad hoc networks: a statistical analysis approach," *Elsevier's Journal of network and computer applications*, 2007.

[16] W. Wang and B. Bhargava, "Visualization of wormholes in sensor networks," in *ACM Workshop on Wireless security*, 2004.

[17] W. Wang and A. Lu, "Interactive wormhole detection and evaluation," *Information Visualization*, 2007.

[18] I. Khalil, S. Bagchi, and N. Shroff, "LITEWORP: a lightweight countermeasure for the wormhole attack in multihop wireless networks," in *IEEE DSN*.

[19] ——, "MobiWorp: Mitigation of the wormhole attack in mobile multihop wireless networks," *Elsevier's Ad Hoc Networks Journal*, 2008.

[20] Y. Wang, Z. Zhang, and J. Wu, "A Distributed Approach for Hidden Wormhole Detection with Neighborhood Information," in *IEEE NAS*, 2010.

[21] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," *Elsevier's AdHoc Networks Journal*, 2003.

[22] Y. Hu, A. Perrig, and D. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *IEEE INFOCOM*.

[23] ——, "Wormhole detection in wireless ad hoc networks," *Department of Computer Science, Rice University, Tech. Rep. TR01-384*, 2002.

[24] S. Čapkun, L. Buttyán, and J. Hubaux, "SECTOR: secure tracking of node encounters in multi-hop wireless networks," in *ACM Workshop on SASN*, 2003.

[25] I. Rhee, M. Shin, S. Hong, K. Lee, and S. Chong, "On the levy-walk nature of human mobility," in *IEEE INFOCOM*, 2008.