RouteDetector: Sensor-based Positioning System that Exploits Spatio-Temporal Regularity of Human Mobility

Takuya Watanabe
Waseda University

Mitsuaki Akiyama
NTT Secure Platform Labs

Tatsuya Mori
Waseda University

Abstract

We developed a novel, proof-of-concept side-channel attack framework called *RouteDetector*, which identifies a route for a train trip by simply reading smart device sensors: an accelerometer, magnetometer, and gyroscope. All these sensors are commonly used by many apps without requiring any permissions. The key technical components of RouteDetector can be summarized as follows. First, by applying a machine-learning technique to the data collected from sensors, RouteDetector detects the activity of a user, i.e., "walking," "in moving vehicle," or "other." Next, it extracts departure/arrival times of vehicles from the sequence of the detected human activities. Finally, by correlating the detected departure/arrival times of the vehicle with timetables/route maps collected from all the railway companies in the rider's country, it identifies potential routes that can be used for a trip. We demonstrate that the strategy is feasible through field experiments and extensive simulation experiments using timetables and route maps for 9,090 railway stations of 172 railway companies.

1 Introduction

Modern smart devices, such as smartphones, smart watches, and smart glasses, have powerful embedded sensors such as accelerometers, magnetometers, gyroscopes, ambient light sensors, and heart rate monitors. While these sensors are used to provide new user experiences, they also bring the new line of side-channel attacks [1, 2, 3, 4, 5, 6, 7, 8].

Let us consider a new side-channel attack called SPS (sensor-based positioning system), which also exploits sensors of smart devices. The ultimate goal of an SPS attack is to estimate the location of a user by reading sensors but without using conventional geolocation methodologies such as GPS, cell tower signals, or WiFi. Clearly, achieving the goal is difficult, primarily due to the high degree of freedom of user mobility.

The goal of this work is to make the SPS attack feasible. To this end, we exploit the *spatio-temporal regularity of human mobility patterns* [9]; e.g., a person may use a fixed route on a transportation system for her/his commuting. Also, vehicles of transportation systems are generally expected to exhibit a temporal regularity unless they encounter operation problems such as natural disasters or rail accidents. We expect that exploiting the regularity enables us to reduce the degree of freedom of human mobility.

With this approach in mind, we develop a novel proof-of-concept attack framework called *RouteDetector*, which targets the location of passengers of transport service. It aims to identify the route of your train trip (i.e., the sequence of train stations) by simply reading three hardware sensors – accelerometer, magnetometer, and gyroscope – which are all accessible from any apps without requiring any permissions. A unique technical concept of RouteDetector is that it makes use of not only data collected from multiple sensors embedded in a smart device, but it also leverages external data that can extract privacy information by correlating with collected sensor data.

The key technical components of RouteDetector can be summarized as follows: First, by applying a machine-learning technique to the data collected from sensors, RouteDetector classifies the activity of a user, e.g., walking, riding on a moving vehicle, or other status such as still. Next, using the sequences of the detected activities, RouteDetector extracts departure/arrival times of vehicle(s). Finally, RouteDetector correlates the extracted departure/arrival times of vehicle(s) with timetables/route maps of all vehicles and searches the potential mobility paths.

The key findings of this work are summarized as follows:

- Our field experiments using smart devices demonstrate that the RouteDetector framework can detect departure/arrival times of vehicles with errors smaller than six seconds on average.

- Our extensive simulation experiments using timetables and route maps for 9,090 railway stations of 172 railway companies demonstrate that given a sequence of departure/arrival times, RouteDetector can identify routes used for a trip by train, and the average number of identified routes becomes close to one if the number of stations used on a trip is more than six.

These findings support that the attack is feasible.

The rest of this paper is organized as follows. Section 2 describes the threat models we assume for *RouteDetector* In section 3, we present the details of the *RouteDetector* framework. Section 4 shows the results of performance evaluation. Section 6 discusses the limitations of *RouteDetector* and future research directions. We also discuss the possible counter measures against *RouteDetector*. Section 7 summarizes the related work. We conclude our work in section 8.

2 Threat models

Our threat model assumes that a malicious software, which requires only a permission of Internet connection, is installed on the victim's device. The software keeps collecting sensor values and estimating the activities of the owner of the device; i.e., walking (running), moving on a vehicle, or other. Sequences of detected activities are periodically sent to the adversary's computer. The adversary's computer estimates the route of transportation by analyzing the sequences. Note that it is also possible that the user device computes the estimation of routes and sends the estimated results to the adversary. It is easy for an adversary to know the hardware model of the smart device; for instance, in the Android platform, by accessing the fields of `Android.os.Build` class, he/she can obtain the hardware information, such as brand, manufacturer, and/or model. He/she can also know whether a smart device is being held in someone's hand or is inside a bag by reading the ambient light sensor or proximity sensor. Because the threat model targets passengers on public transportation systems, it is not useful where no public transportation system is available. We also assume that the adversary knows the list of public transportation systems that would likely be used by the victim. For instance, if a victim lives in a particular country, the adversary assumes that the victim may use any of railways available in that country. We also need to assume that the transportation system operates punctually; otherwise, RouteDetector's estimation may be inaccurate. We will study the issue in Sec. 4. Other limitations will be discussed in Sec. 6.

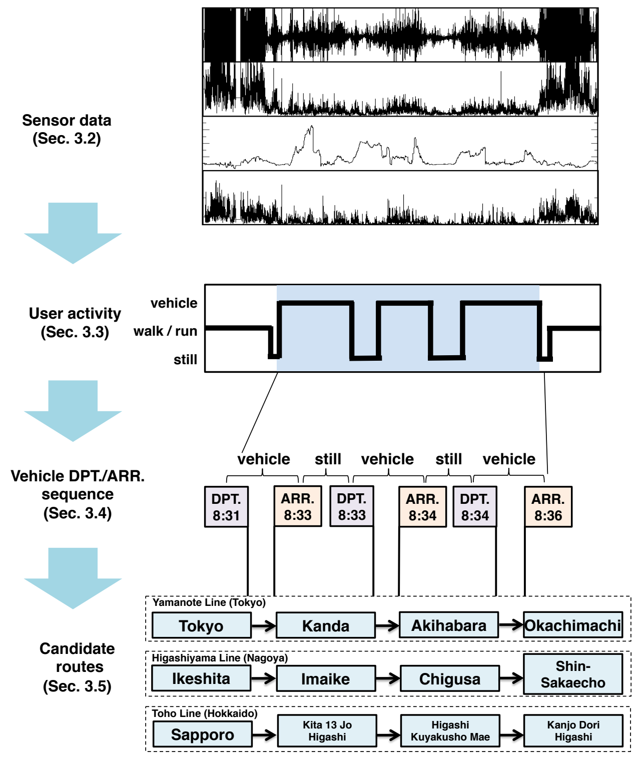

Figure 1: High-level overview of the RouteDetector framework.

3 RouteDetector Framework

In this section, we present an overview of the *RouteDetector* framework (Sec. 3.1). Then, we describe the sensors we used for our analysis (Sec. 3.2). We then describe the key technical components of the *RouteDetector* framework; the detection of user activities in Sec. 3.3, detection of departure/arrive time sequences of vehicles in Sec. 3.4, and the extraction of candidate routes in Sec. 3.5.

3.1 Goal and Overview

The goal of the *RouteDetector* framework is to identify the route of a vehicle used by an owner of a smart device by reading the device's sensors. If a vehicle is a passenger train, a route is defined as a set of stations along a path. Figure 1 depicts the high-level overview that achieves the goal, together with the number of corresponding subsections that describe the technical details.

First, it reads values from sensors. As sensors, we picked up accelerometer, linear acceleration, magnetometer, and rotation vector, which are all accessible from any app without requiring any permissions. Details of data collection are described in Sec. 3.2. Next, we ex-

Table 1: Summary of sensors.

Sensor	Type	unit	Permission	Description
accelerometer	physical	m/s^2	Not required	Acceleration applied to a device including the gravity.
linear acceleration	virtual	m/s^2	Not required	Acceleration applied to a device excluding the gravity.
magnetometer	physical	μT	Not required	Strength of geomagnetic field.
gyroscope	physical	rad/s	Not required	A device's rate of rotation.

tract user activities from the collected sensor data. The user activities are defined as a set of three classes, *walking*, riding on a moving vehicle (*vehicle* in short), and *others*, which includes various activities such as standing, sitting, or sleeping. To this end, we pre-process raw sensor data so that we can apply a supervised machine-learning (ML) approach. As a supervised ML algorithm, we adopt random forest, which is known to achieve robust and good performance for multi-class classification tasks. Details of data pre-processing and ML application are described in Sec. 3.3. From the extracted user activities, we can identify sequences of vehicle departure/arrival times. For instance, if we find a consecutive pairs of *vehicle* and *others*, it is likely that a user was on a vehicle. We can also consider cases in which a user made a transit. Details of detecting vehicle departure/arrival time sequence are described in Sec. 3.4. Finally, from an extracted vehicle departure/arrival time sequence, we search candidate routes, using timetables and railway route maps that cover the potential residential area of the victim, e.g., a country. We develop a fast algorithm that works in a breadth-first search manner. Details of extracting departure/arrival time sequence are described in Sec. 3.4.

3.2 Sensor Data

Of the available sensors embedded into a smart device, we adopt four sensors; accelerometer, linear acceleration, magnetometer, and rotation vector. Table 1 summarizes the sensors we used. Although we tested other sensors, such as an ambient light sensor, the data was not effective in detecting user activities. Note that the four sensors can be divided into two classes: physical sensors and virtual sensors. While the accelerometer, magnetometer, and gyroscope are physical sensors that read raw values, the remaining sensor, linear acceleration, is a virtual sensor whose values are computed based on physical sensors. We note that the sensors are accessible from any app without requiring any permissions; therefore, they are prone to be covertly abused by a malicious developer.

We developed an Android app that collects the sensor data. All the values are collected at a rate of 10 Hz, i.e., read 10 values per second. The app also has a function to generate labels that are used for supervised ML.

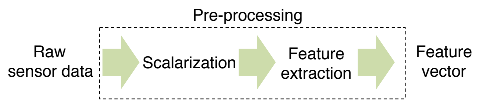

Figure 2: Overview of data pre-processing.

3.3 Detection of User Activities

Using the collected sensor data, we classify user activities into three distinct classes, *walk*, *vehicle*, and *others*. Note that *vehicle* refers to the status when a user is on a moving vehicle. If n user is standing on a vehicle, which is stopping at a station, his/her status is likely classified as *others*. We first pre-process raw sensor data in Sec. 3.3.1. Next, we apply a supervised machine-learning (ML) approach to the pre-processed data to detect user activities in Sec. 3.3.2.

3.3.1 Data Pre-processing

We apply several data pre-processing techniques to the raw sensor data. Figure 2 summarizes the data pre-processing scheme. First, to eliminate the effect of differences in the directions in 3D space, we compute a norm for each 3D vector; i.e., $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$. Figures 3 (a) and (b) are examples of scalarized data. We then divide time series data into a set of blocks. A block consists of N samples for each sensor data; i.e., for each sensor data, a block b_i has data: $\mathbf{D}^{(i)}(a) = \{a_1^{(i)}, a_2^{(i)}, \ldots, a_N^{(i)}\}$. We experimentally set N as $N = 20$, which corresponds to 2 seconds length with the 10-Hz rate of sensor data sampling. For each block, we extract features that can be used to characterize the patterns of temporal variability for the three classes. To this end, we adopted simple metrics; i.e., mean, standard deviation, minimum, and maximum. Finally, we normalize the data by subtracting means and dividing by standard deviations. In summary, the time series data is divided into blocks, and each block consists of four features for four sensors, resulting in feature vectors with $4 \times 4 = 16$ dimensions.

3.3.2 Classifying User Activities

Using the pre-processed sensor data, we classify activities into three classes; *walk*, *vehicle*, and *others*. As a classification scheme, we adopt the Random forest algorithm, which is an ensemble learning algorithm used for classification or regression. In the training phase, the Random forest algorithm constructs multiple decision trees using randomly sampled data. In the classification phase, it predicts the most plausible class by taking the majority votes of the multiple decision trees. The good feature of Random forest is that it naturally achieves multi-class classification with a measure of score. We note that we also tested other supervised machine learning algorithms, such as SVM or logistic regression. It turned out that the differences in performance among the algorithms were not significant, but the Random forest algorithm worked best.

3.4 Detection of Departure/Arrival Time Sequences of Vehicles

Using the detected user activities, we extract sequences of vehicle departure/arrival times. Among the user activities, we are most interested in *vehicle* activity because the start/end of the activity corresponds with the departure/arrival, respectively. However, as shown in Fig. 3 (c), the predicted activities include some noise due to the inevitable classification errors. To reduce the effect of classification errors, we leverage the temporal correlation of the activities; i.e., once a user gets on a vehicle, it is likely that he/she stays on the vehicle for several minutes. Namely, we use the exponentially weighted moving average (EWMA) to account for temporal correlation of data.

Let A_n be the classified activity at block n, and $\mathcal{W}$, $\mathcal{V}$, and $\mathcal{O}$ be the set of blocks that are classified as *walk*, *vehicle*, and *others*, respectively. We define W_n, V_n, and O_n as

$$\begin{aligned} W_n &= \mathbf{1}_{\mathcal{W}}(A_n) \\ V_n &= \mathbf{1}_{\mathcal{V}}(A_n) \\ O_n &= \mathbf{1}_{\mathcal{O}}(A_n), \end{aligned}$$

where $\mathbf{1}_Y(x)$ is an indicator function that is defined as

$$\mathbf{1}_Y(x) = \begin{cases} 1 & \text{if } x \in Y \\ 0 & \text{if } x \notin Y. \end{cases}$$

First, we compute the EWMA of V_n; i.e.,

$$\overline{V_n} = \lambda V_n + (1-\lambda)\overline{V_{n-1}},$$

where $\overline{V_n}$ is EWMA and $0 \leq \lambda \leq 1$ is a constant parameter that determines the smoothing factor. If λ is close to one/zero, the EWMA has a larger weight on the last observation/past observations. The parameter λ is empirically configured, as we will show later. Although the EWMA introduces a certain time lag to the original data, the size of the lag was negligible, as we will show later. Using the EWMA, the classified activities are corrected, as

$$\widehat{V_n} = \begin{cases} 1 & \text{if } \overline{V_n} \geq 0.5 \\ 0 & \text{if } \overline{V_n} < 0.5. \end{cases}$$

Figure 3 (d) shows smoothened user activities with the EWMA.

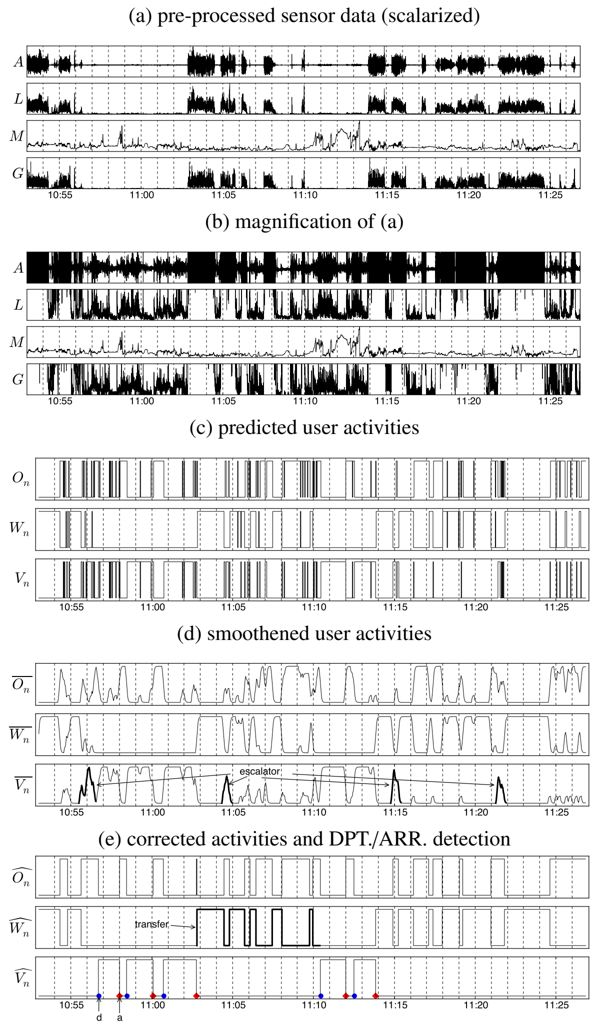

Figure 3: (a): pre-processed sensor data, (b) magnification of (a) in Y-axis, (c) predicted user activities, (d) smoothened user activities, and (e) corrected user activities and departure/arrival times. In panels (a) and (b), A, L, M, and G represents accelerometer, linear acceleration, magnetometer, and gyroscope, respectively. In panel (e), circles/squares are detected departure/arrival times, respectively.

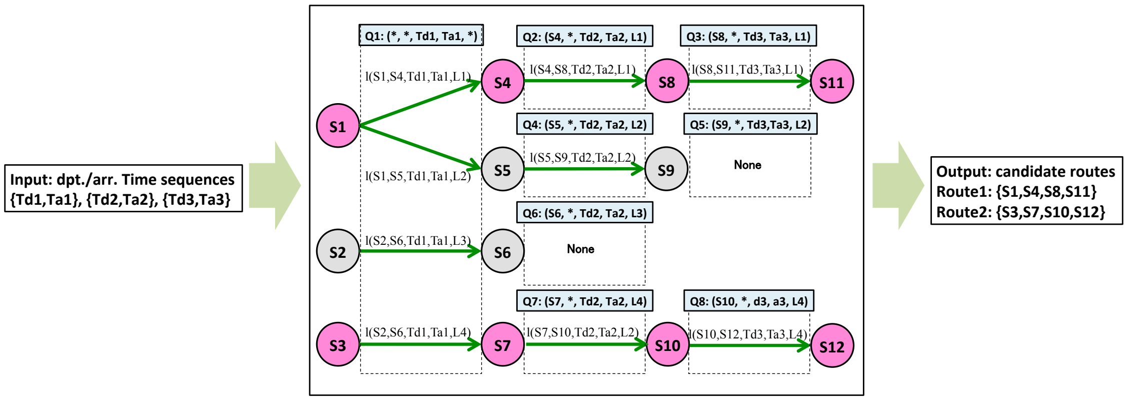

Figure 4: Diagram of the route detection algorithm.

Next, using the corrected activities $\widehat{V_n}$, we extract departure/arrival time sequences using the following algorithm, where τ is a threshold that determines the minimum length of time for a trip between two stations. In this calculation, we set $\tau = 60$ (seconds).

Algorithm 1 Vehicle DEP./ARR. time sequences detection algorithm.

```
 1: $D$ =**false**                                          ▷ Initial state
 2: **for all** $n = 1,2,\ldots$ **do**
 3:     **if** $\widehat{V_n} = 0$ AND $\widehat{V_{n+1}} = 1$ **then**
 4:         $T_d = t_{n+1}$                           ▷ $t_n$ is time at block $n$.
 5:         $D$ =**true**                  ▷ A vehicle has been departured.
 6:     **if** $\widehat{V_n} = 1$ AND $\widehat{V_{n+1}} = 0$ AND $D = 1$ **then**
 7:         $T_a = t_{n+1}$
 8:         $D$ =**false**
 9:         **if** $T_a - T_d > \tau$ **then**
10:             **return** $T_a, T_d$
```

We note that using blocks that were not classified as *vehicles*, i.e., $\{n; \widehat{V_n} = 0\}$, W_n and O_n can be corrected using the similar procedure. Tracking W_n and O_n is useful for detecting transferring lines; i.e., if we observe a sequence of classified activities such as *vehicle* (3 mins), *walk* (2 mins), *others* (4 mins), and *vehicle* (5 mins), it is likely that a person changed lines. Figure 3 (e) shows such an example. The victim first got on a train and got off the train after three stations. He/she then changed lines (see the area "transfer" shown in the graph of $\widehat{W_n}$), and got on the next train.

As we shall see later, the activity of riding an escalator could be misclassified as being on a *vehicle*, although a person may be using it for transferring lines. Such a misclassification can be safely removed with this heuristic. Figure 3 (d) and (e) show such an example where all the ground-truth escalator points, which were misclassified as "vehicle" by random forest, are successfully eliminated in the corrected user activities. The heuristics are also useful for eliminating other errors regarding activity detection.

3.5 Extracting Candidate Routes

Finally, using the extracted sequences of departure/arrival times, we estimate candidate routes. We formulate the estimation task as follows. Using railway route maps, we first create a single graph that consists of nodes (stations) connected by links (railroads). Next, using timetables corresponding to the railway route maps, we extend the graph so that it expresses temporal structure. Let us call the extended graph a "train graph." In a train graph, a link $l(A,B,T_d,T_a,L)$ expresses a vehicle that departures station A at time T_d and arrives at station B at time T_a; A and B are adjacent stations on line L. Note that we do not need to build/keep an entire train graph beforehand. Instead, we compile a set of all links and dynamically build subgraphs by applying our search algorithm to the set of links.

We use Fig. 4 to demonstrate how the algorithm of searching candidate routes works. In the example, we have the input departure/arrival time sequence of $\{T_d j, T_a j\}$ $(j = 1,2,3)$. Given the input, we first extract a set of links that satisfies $l(*,*,T_d 1,T_a 1,*)$ (Q1: query 1). In the example, we found four links; $(S1,S4,T_d 1,T_a 1,L1)$, $(S1,S5,T_d 1,T_a 1,L2)$, $(S2,S6,T_d 1,T_a 1,L3)$, and $(S3,S7,T_d 1,T_a 1,L4)$. For each link above, we recursively search the succeeding links. For instance, to find a link (vehicle) that departs station $S4$ at time $T_d 2$ and arrives at station X at time $T_a 2$ on line $L1$, we search a link that satisfies $l(S4,*,T_d 1,T_a 1,L1)$ (see Q2) and found $S8$ is the destination station. If we do not find any links that satisfy the given condition, we remove the paths from the search (see Q5, Q6). By continuing the above procedure, we can enumerate paths that satisfy the input departure/arrival time sequences; i.e., routes $\{S1,S4,S8,S11\}$

Table 2: Smart devices used for our analysis.

Device name (abbreviation)	Type	OS
HTC J Butterfly (HTC)	Smartphone	Android 4.1.1
Nexus 7 (Nexus)	Smart Tablet	Android 4.4.4

and $\{S3, S7, S10, S12\}$ in the example.

Finally, when we get multiple routes for a given time sequence, it is useful that we can sort them according to some metrics. To this end, we compute the popularity of routes, as follows: For each link consisting of a route, we compute the number of other links that share the same pair of origin/destination stations with that link. We then sum up the numbers along the links of a route and define the result as a score. If a route has a larger score, it means that a larger number of trains run on that route. We adopt this score as a metric that expresses the popularity of a route.

4 Evaluation

In this section, we evaluate the performance of the RouteDetector framework. We first summarize the datasets we used for our analysis. Second, we evaluate the accuracy of the user activities detection scheme. We then evaluate the accuracy of departure/arrival time sequence detection. Finally, we evaluate the effectiveness of the candidate routes detection scheme.

4.1 Data

The data we collected for evaluation is broadly classified into two datasets. The first set consists of sensor data used for detecting departure/arrival time sequences. The second set consists of timetables and railway route maps that are used for building a train map, which is then used to search candidate routes for a given time sequence.

4.1.1 Sensor Data

Table 2 presents the two smart devices used for our analysis. As we shall see later, different hardware sensors generally exhibit different values when given the same input. Therefore, we need to train each classification model for each device. Details regarding to the differences in device hardware will be discussed in Section 6.

Table 3 summarizes the sensor data we collected. These data were measured across seven lines, operated by two railway companies. Four lines, Yamanote Line, Chuo Line, Keihin-Tohoku Line, and Saikyo Line, are operated by East Japan railway company. Three subway lines, Fukutoshin Line, Marunouchi Line, and Nanboku Line are operated by Tokyo Metro. Of these lines, Yamanote Line is one of the busiest and most important lines that connect major stations in Tokyo. As shown in the table, we distinguish between two measurement types: a device held by hand (H) or located inside a still bag (B), which could be placed on the knee or on a rack. As we mentioned in Section 2, an adversary can distinguish the hardware of devices. He/she can also know whether a smart device is being held in someone's hand or is inside a bag by reading the ambient light sensor or proximity sensor.

Table 3: Sensor data collected for our analysis.

Data name	Device	Type	# stations	# lines	# blocks
HTC_H	HTC	H	57	5	12,007
HTC_B	HTC	B	29	1	2,561
Nexus_H	Nexus	H	29	1	2,543
Nexus_B	Nexus	B	54	5	8,576

Table 4: Statistics of the train map built from railway route maps and timetables. Number of links is taken from timetables for weekdays.

# railway companies	# lines	# stations	# links
172	597	9,090	2,277,397

4.1.2 Railway Route Maps and Timetables

While the coverage of collected sensor data is limited to a certain location, we use entire train services operated in Japan for building a train map. Table 4 summarizes the data we collected. Note that a link $l(A, B, T_d, T_a, L)$ is defined in Section 3.5. We also note that if we can further specify the residential location of a victime, e.g., Kyoto area, the amount of data and candidate routes can be further reduced.

4.2 User activities detection

We applied our user activities detection scheme to the data shown in Table 5. The parameters of random forest were empirically optimized as $n = 50$ and $m = 4$, where n is the number of trees and m is the number of features used for each tree. To assess the generalization of the result, we employed 10-times, 10-fold cross-validation tests. We focused on the accuracy of detecting vehicles because it plays a crucial role in determining the departure/arrival time sequence. If a block of *vehicle* was incorrectly classified as *walk* or *others*, we defined it as a false negative. If a block of *walk* or *others* was classified as *vehicle*, we define it was false positive.

Table 5: Numbers of labeled blocks used for evaluating performance of activity detection. All the labeled blocks are collected at the stations of Yamanote Line.

Data	*vehicle*	*walk*	*others*
HTC_H	609	1,327	510
HTC_B	691	1,360	510
Nexus_H	686	1,352	505
Nexus_B	602	1,304	505

Table 6: Performance of detecting *vehicle* activity. ACC, FNR, and FPR are accuracy, false negative rate, and false positive rate, respectively.

Data	ACC (mean/std)	FNR (mean/std)	FPR (mean/std)
HTC_H	0.941/0.011	0.042/0.022	0.078/0.013
HTC_B	0.965/0.009	0.024/0.012	0.047/0.014
Nexus_H	0.943/0.013	0.041/0.014	0.074/0.021
Nexus_B	0.969/0.009	0.023/0.012	0.041/0.016

Table 6 summarizes the results. We noticed that classification accuracies are generally good in all the cases. We also noticed that measurement types of H, i.e., a device was inside a still bag, gave better accuracies. The result is intuitively natural because holding a smart device by hand may introduce motion noise.

4.3 Departure/Arrival Time Sequences Detection

Next, we applied our departure/arrival time sequence detection algorithm to the extracted user activities. For each dataset, we picked up departure/arrival time sequences of 30 stations. The 30 samples are divided into a training set and a test set. Using the training set, the parameter of EWMA, λ, was optimized so that the difference between the detected departure/arrival time and observed departure/arrival time is minimized. Note that "detected" times are derived from sensors, "observed" times are manually labeled ones, and "scheduled" times are derived from a timetable corresponding to a train. To evaluate the performance, we employed 10-times, 3-fold cross-validation tests; i.e., 30 samples are randomly divided into 20 samples for a training set and 10 samples for a testing set, using different random seeds. Table 7 summarizes the absolute errors between detected and observed departure/arrival times. Note that observed departure/arrival times are not necessarily the scheduled times listed in timetables. The difference between the observed and scheduled times is shown in Fig. 5.

As we see, the detected departure/arrival times are close to the observed departure/arrival times. Maximal

Table 7: Absolute errors between detected times and observed (ground truth) times; departure (top) and arrival (bottom). m and σ are mean and standard deviation, respectively.

absolute errors of detected departure times.				
Data	min (sec)	max (sec)	m (sec)	σ
HTC_H	1.97	3.54	2.79	0.46
HTC_B	2.04	3.06	2.53	0.23
Nexus_H	2.33	7.94	4.60	1.84
Nexus_B	1.55	2.76	2.17	0.24
absolute errors of detected arrival times.				
Data	min (sec)	max (sec)	m (sec)	σ
HTC_H	2.52	6.75	4.13	1.18
HTC_B	1.71	4.63	3.21	0.77
Nexus_H	3.07	10.78	6.03	2.22
Nexus_B	2.22	5.16	3.43	0.80

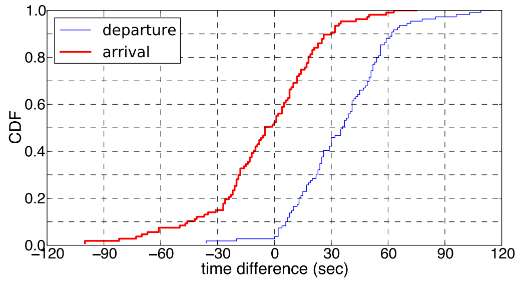

Figure 5: Distributions of difference between observed and scheduled times. Departure times (top) and arrival times (bottom).

time differences are less than 3-11 seconds. Time differences are within 3-7 seconds. Note that all the departure/arrival events are perfectly detected. In addition, the observed departure/arrival times are also close to the scheduled times. Roughly 85% of trains depart within 60 seconds after the scheduled time has passed. Roughly 75% of trains arrived within 30 seconds around the scheduled time.

In summary, the detected departure/arrival times by the RouteDetector framework are close to the observed departure/arrival times, which are close to the scheduled times. In the next subsection, we show how we search routes given the detected departure/arrival time sequences. We also present several case studies in Sec. 5.

4.4 Candidate Routes Detection

While the evaluation of departure/arrival time detection scheme required empirical data, the evaluation of the candidate routes detection algorithm can be generalized

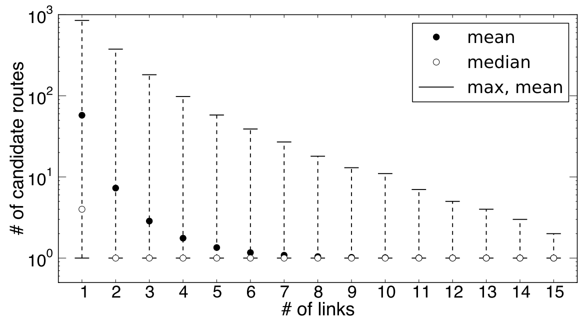

Figure 6: Number of links vs. number of candidate routes.

by exploring paths on a train graph[1]. Using the train graph constructed from the data shown in Table 4, we study the relationship between the number of links and the number of corresponding candidate routes. Figure 6 shows the results. We can see that average number of identified routes becomes close to one if the number of stations used on a trip is more than six; i.e., if we observe more stations, the sequence of departure/arrival times become more unique. Even if the number of links is one, roughly 50% of time sequences T_d, T_a have less than four candidate routes.

Next, we study how quickly the search algorithm works. From the entire train graph, we first enumerate the routes whose lengths are less than 15 links, where we allowed, at most, two line changes. The number of enumerated routes was 6,404,455,757. Using the C++ implementation of the algorithm that runs on a commodity PC, all these routes were searched within 74 mins. On average, a route was searched within 7.1 microseconds. Thus, the candidate routes detection worked quickly even though the scale of the train graph was huge.

5 Case study

In this section, we demonstrate the feasibility of the RouteDetector framework through the field experiments. Using sensor data collected from smartphone or tablet, we try to identify a route used for a trip. For brevity, we present three typical cases below. Figure 7 presents a map of lines used for the case study.

Case 1 In this case, the train trip involved two lines, Yamanote line and Marunouchi line as shown in Fig. 7. Figure 3 presents the measured/derived data for the case 1. From Fig. 3 (e), we detected departure/arrival time

[1]Because enumerating all the possible paths on a train graph could cause an explosion of states, we limit our search to the paths with lengths less than 15 stations.

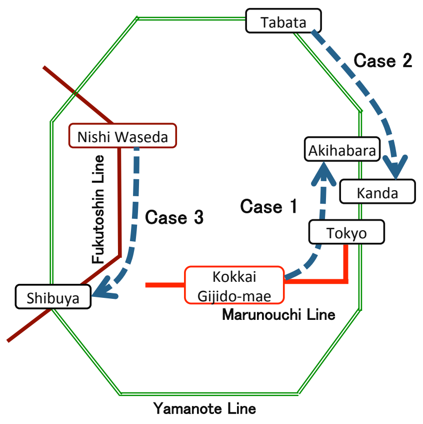

Figure 7: Map of lines used for case study analysis.

Table 8: Detected/observed/scheduled times for case 1. Detected and observed times are rounded.

activities	detected	observed	scheduled
walking etc.		–	
departure arrival	10:56 10:58	10:56 10:58	10:56 10:58
departure arrival	10:58 11:00	10:58 11:00	10:58 11:00
departure arrival	11:00 11:03	11:00 11:03	11:00 11:03
walking etc.		–	
departure arrival	11:10 11:12	11:10 11:12	11:10 11:12
departure arrival	11:12 11:14	11:12 11:14	11:12 11:14
walking etc.		–	

sequence. The results are summarized in Table 8. As we see, all the detected departure/arrival times were correctly detected. Next, given this time sequence, we search the corresponding routes. The result is shown in Table 9, which shows two routes are identified. Of the identified two routes, the route #1 had higher score and was identical to the ground truth. Thus, the RouteDetector successfully detected a route used for a train trip from sensor data.

Case 2 The case 2 was measured at Yamanote line. There was no transferring lines. The origin/destination stations were Tabata station and Kanda station, respectively. The trip involved 8 stations. Figure 8 presents the detected activities and departure/arrival time sequence. In this case, the detected departure/arrival times were correctly detected. Given the time sequence, a unique route was identified. The identified route was identical to the ground truth.

Table 9: Two identified routes for case 1.

No.	ground truth	route #1	route #2
1	Kokkai-gijido-mae	Kokkai-gijido	Edogawabashi
2	Kasumigaseki	Kasumigaseki	Gokokuji
3	Ginza	Ginza	Higashi Ikebukuro
4	Tokyo	Tokyo	Ikebukuro
transfer			
4	Tokyo	Tokyo	Ikebukuro
5	Kanda	Kanda	Kanamecho
6	Akihabara	Akihabara	Sengawa
score	–	2,664	2,277

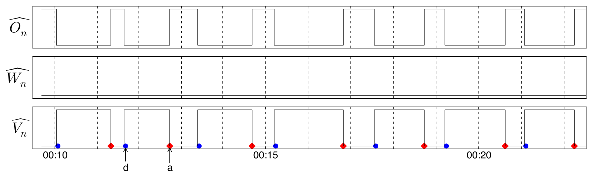

Figure 8: Detected activities of the case 2.

Case 3 The case 3 was measured at Fukutoshin Line. Again, there was no transferring lines. The origin/destination stations were Nishi Waseda station and Shibuya station, respectively. In this case, while the detected departure/arrival times were identical to the observed times, they were slightly different from the scheduled time; i.e., the train was delayed at the time of measurement. We will discuss the issue of train operation in the next section. Given the detected time sequence, no train route was identified from the train graph.

6 Discussion

In this section, we discuss several limitations of the RouteDetector framework. We also discuss countermeasures against the new threat brought by the RouteDetector framework.

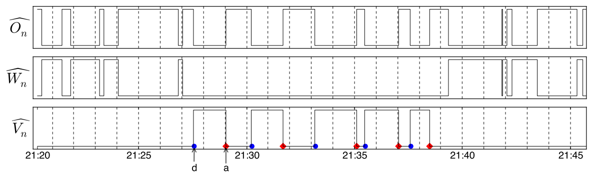

Figure 9: Detected activities of the case 3.

Table 10: Detected/observed/scheduled times for case 3.

activities	detected	observed	scheduled
walking etc.	–		
departure	21:27	21:27	21:26
arrival	21:29	21:29	21:28
departure	21:30	21:30	21:28
arrival	21:32	21:32	21:32
departure	21:33	21:33	21:32
arrival	21:35	21:35	21:35
departure	21:35	21:35	21:35
arrival	21:37	21:37	21:37
departure	21:37	21:37	21:37
arrival	21:39	21:39	21:39
walking etc.	–		

6.1 Limitations

Cross-Device Differences Our thread model assumes that an adversary knows the type of hardware to be attacked; i.e., he/she needs to have training data for detecting user activities for each device. In fact, we found that a random forest classifier trained to work with smartphone data did not work well for detecting the activities of tablet users. This observation suggests that a difference in hardware sensors is sensitive to the user activity detection scheme. One approach to this problem is to prepare training models for various devices. Another possible approach is to apply some data-processing techniques that can absorb the differences in the measurements of sensor values. We leave the issue for our future work.

Types of Vehicles While the target of this work was passenger trains, there are other types of transportation services, such as monorails or airplanes. If we can assume that vehicles are operated accurately according to timetable schedules, we may have a good chance to detect a route used for a trip. We conjecture that the RouteDetector will not work well for automobile transport services such as public bus transportation because of large deviations in operation timeline.

Train Operation Clearly, the success of the RouteDetector framework relies on the accuracy of the train operation. The detection accuracy may be limited in an environment where many trains tend to be delayed. For such a case, we need to study up to what amount of delay the attack works. To this end, we could artificially add a random delay and see how the framework reacts. We leave the analysis for our future work. We note that even in case of delay, some transportation systems provide information in real-time. Such information could be used to make the system more tolerant to delay.

We also note that by continuously targeting a victim, an adversary can obtain multiple observations, which likely include the correct estimations; e.g., commuting routes. Thus, by collecting many candidate routes used by a target, an adversary can figure out locations frequently visited by the target in a statistical way.

6.2 Countermeasures

Let us discuss some ways to mitigate or eliminate the risk caused by the attacks using the RouteDetector framework. Michalevsky et al., presented *Gyrophone* [6], which is an attack that recognizes speech by reading gyroscope. They mentioned countermeasures in their paper that apply low-pass filtering to the raw samples provided by sensors. If certain pass frequencies are enough for most of the applications, the filtering can be done without negative effects. In addition, they mentioned that it should be controlled by permission mechanisms or certain explicit authorization by the user when certain applications require an unusually high sampling rate. In the same way, restricting access to raw sensor data and building some filtration mechanism that can remove sensitive information without sacrificing other functions would be promising approaches as countermeasures against the attack with RouteDetector. For instance, to build a pedometer app, a developer can use a specific API that can retrieve step counts, instead of reading row sensor values of accelerometer. Thus, building wrapper APIs that provide many useful functions, while hiding raw data, is a promising approach to thwart sensor-based side-channel attacks.

7 Related work

Techniques of sensor data analysis on mobile devices are mainly used for extending the range of application of mobile services, e.g., activity recognition and location-based services. On the contrary, attackers can expose user's privacy by using above similar techniques analyzing sensor data. We introduce techniques for both benign and malicious uses.

Positioning without GPS An indoor positioning system (IPS) is presented as a solution to detect/navigate objects or people inside a building [10]. Instead of using GPS, IPS techniques make use of other information sources such as radio wave, acoustic signals, and optical signals. As an example of malicious use of the positioning technique, Michalevsky et al. demonstrated that their developed *PowerSpy* application enables the attacker to infer the target device's location over those routes or areas by simply analyzing the target device's power consumption [8].

Device fingerprinting A device fingerprinting is other positive usage of sensors to identify and authenticate physical devices. Many studies reported that various IDs on a smartphone, e.g., IMEI (device ID), are easily stolen by malicious apps. To thwart ID-theft, Dey presented *AccelPrint*, which is a system that fingerprints based on the accelerometer, in order to identify devices without any specific ID or cookie [5]. Das et al. also discussed the feasibility of using sensors embedded in smartphones, i.e., microphones and speakers, to uniquely identify individual devices [7].

Activity Recognition The *CenceMe* system developed by Miluzzo et al. [11] combines the inference of individuals' activity using sensors' information with sharing of it through social networking services. To classify activities (sitting, standing, walking, running) of individuals, the preprocessor of CenceMe calculates the mean, standard deviation, and number of peaks of the accelerometer readings along the three axes of the accelerometer. RouteDetector's activity detection scheme is similar to this one, but it is extended to capture the motion of vehicles. RouteDetector also uses other hardware sensors, such as a magnetometer and gyroscope, which also play a key role in improving detection accuracy.

The accelerometer sensor provides an attacker with other opportunities to build new attacks. Many attacks targeting motion sensors, i.e., accelerometers and gyroscopes, that are embedded in smartphones are inferring user inputs, e.g., passwords on touch-screens by monitoring readings collected from motion sensors [1, 2, 3, 4].

Sensor Access Control Although various kinds of sensor information contribute to extend and improve mobile computing and services, privacy issues have already been exposed as mentioned above. One of the most practical defenses is access control to sensor data. Unnecessary access by apps to sensor data should be controlled by OS or middleware on a device. *FlaskDroid* [12] and *ipShield* [13] are implemented as middleware on Android OS and provide fine-grain access control mechanism to resources including sensor information.

8 Conclusion

A novel, proof-of-concept side-channel attack framework called *RouteDetector* was introduced. The key idea behind the framework is to leverage *spatio-temporal regularity* of human mobility; i.e., we targeted passengers of train systems. Our field experiments demonstrated that the RouteDetector framework detected departure/arrival times of vehicles with errors less than 6 seconds on average. Our extensive simulation experiments using timetables and route maps for 9,090 railway stations of 172 railway companies demonstrated that the *RouteDetector* successfully identified routes used for a trip by train, and the average number of identified routes became close to one if the number of stations used on a trip was more than six. These results quantitatively support that the attack is feasible.

Acknowledgements

A part of this work was supported by JSPS Grant-in-Aid for Challenging Exploratory Research (KAKENHI), Grant number 15K12038.

References

[1] L. Cai and H. Chen, "TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion," in *The 6th USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[2] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: Password Inference using Accelerometers on Smartphones," in *The Twelfth Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2012.

[3] Z. Xu, K. Bai, and S. Zhu, "TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors," in *The fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 2012.

[4] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "TapPrints: Your Finger Taps Have Fingerprints," in *The 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[5] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable," in *The 2014 Network and Distributed System Security (NDSS) Symposium*, 2014.

[6] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing Speech from Gyroscope Signals," in *The 23rd USENIX Security Symposium*, 2014.

[7] A. Das, N. Borisov, and M. Caesar, "Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components," in *The 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[8] Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh, "Powerspy: Location tracking using mobile device power analysis," *CoRR*, vol. abs/1502.03182, 2015.

[9] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, pp. 779–782, June 2008.

[10] Y. Gu, A. Lo, and I. Niemegeers, "A Survey of Indoor Positioning Systems for Wireless Personal Networks," in *IEEE Communications Surveys & Tutorials*, 2009.

[11] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application," in *The 6th ACM conference on Embedded network sensor systems (SenSys)*, 2008.

[12] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies," in *The 22nd USENIX Security Symposium*, 2013.

[13] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Millar, and M. Srivastava, "ipShield: A Framework For Enforcing Context-Aware Privacy," in *The 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.