MetroEye: Smart Tracking Your Metro Trips Underground

Weixi Gu
Tsinghua-Berkeley Shenzhen Institute
Tsinghua University
guweixigavin@gmail.com

Ming Jin
Department of EECS
University of California, Berkeley
jinming@berkeley.edu

Zimu Zhou
Computer Engineering and Networks Laboratory
ETH Zurich
zimu.zhou@tik.ee.ethz.ch

Costas J. Spanos
Department of EECS
University of California, Berkeley
spanos@berkeley.edu

Lin Zhang
Tsinghua-Berkeley Shenzhen Institute
Tsinghua University
linzhang@tsinghua.edu.cn

ABSTRACT

Metro has become the first choice of traveling for tourists and citizens in metropolis due to its efficiency and convenience. Yet passengers have to rely on metro broadcasts to know their locations because popular localization services (*e.g.* GPS and wireless localization technologies) are often inaccessible underground. To this end, we propose *MetroEye*, an intelligent smartphone-based tracking system for metro passengers underground. *MetroEye* leverages low-power sensors embedded in modern smartphones to record ambient contextual features, and infers the state of passengers (*Stop*, *Running*, and *Interchange*) during an entire metro trip using a Conditional Random Field (CRF) model. *MetroEye* further provides arrival alarm services based on individual passenger state, and aggregates crowdsourced interchange durations to guide passengers for intelligent metro trip planning. Experimental results within 6 months across over 14 subway trains in 3 major cities demonstrate that *MetroEye* yields an overall accuracy of 80.5% outperforming the state-of-the-art.

CCS Concepts

•Human-centered computing → Ubiquitous and mobile computing systems and tools;

Keywords

underground public transport; location-based service; smartphone; crowdsourcing

1. INTRODUCTION

With the speed up of urbanization, an increasing number of citizens and tourists choose to commute by metro for its efficiency and convenience [26]. However, metro trips can also bring frustrating experience to passengers due to its semi-closed environment. As GPS and WiFi are usually inaccessible underground, first-time travellers usually have to rely on broadcasts to keep track of their trips. Even for citizens who have spent years in the city, it is not uncommon to miss the station to get off, especially when they are absorbed in reading, calling, chatting, listening to music or playing games. Furthermore, passengers may miss a metro and have to replan their routes if the interchange time is unexpected long during rush hours. A location-aware service to track passengers' status during an entire metro trip, therefore, is urgently needed.

While metro offices often distribute metro timetables to the public, the actually arrival and departure time are subject to random factors, *e.g.*, over-crowdedness, and the timetable can be simply unavailable in some cities [24, 6, 27]. Thus it is unreliable to infer the locations of passengers based on metro timetables. Some mobile applications retrieve the actual metro position from the dispatching system of metro centers. While this approach is accurate, it may incur latency when wireless communication is weak or when the number of users sharply increases.

The rich built-in sensor on modern smartphones offers a promising alternative. Pioneer works exploited various phone-embedded sensors such as accelerations [29] and magnetic field [16] to identify metro stops. The key idea is to detect the jolts and magnetic changes during a metro's motion [14]. By counting the numbers of stops at stations, they hold potential to track passengers without the need of GPS or wireless-based localization techniques. However, they only consider the spatial states of metros yet ignore the inner temporal relations, and fail to distinguish stops *between* stations and *at* stations, which may lead to false reminders for passengers. Other works [28, 25] combine both inertial sensors and a metro timetable into a Hidden Markov Model (HMM) [28], and filter in-between stops based on the timetable. While these works take the temporal status of metros into account using a timetable and temporal sequential models, a fixed timetable is not always available in those areas with large passenger flow (*e.g.* China, India), thus limiting the applicability of these schemes to certain countries (*e.g.* Japan, Switzerland). More importantly, previous works [29, 16, 14, 28, 25] focuses on the *Stop* and *Running* states of a *single* metro. None of them has considered a joint trip of *multiple* metros, and the *Interchange* state between metros, which is an important yet missing factor in metro passenger tracking and metro trip optimization. For instance, it can be annoying and inconvenient for passengers to retype the destination of another metro when hurrying to the platform of the next metro during interchange. And interchange time statistics can facilitate metro officials optimize metro schedules and help passengers plan further trips to avoid crowded interchange stations.

In this paper, we propose *MetroEye*, an intelligent smartphone-based metro passenger tracker. Unlike previous works [29, 16, 14,

MOBIQUITOUS '16, November 28-December 01, 2016, Hiroshima, Japan

DOI: http://dx.doi.org/10.1145/2994374.2994381

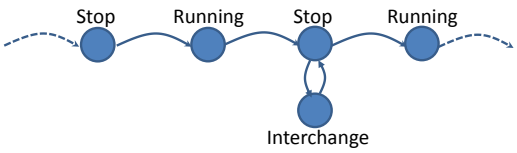

Figure 1: The diagram of a metro trip process

28, 25] that track the *Stop* and *Running* of *a single metro*, *MetroEye* is the first-of-its-kind that jointly tracks the *Stop*, *Running*, and *Interchange* of passengers during *a whole metro trip*. As shown in Figure 1, a metro trip consists of a series of *Stop* (passenger on a metro halting at a station), *Running* (passenger on a running metro), and *Interchange* (passenger waiting or transferring at an interchange station) states. *MetroEye* seamlessly tracks the change of these three states by invoking the low-power sensors to monitor ambient magnetic field, acceleration, and cell tower signal strength. It also carefully extracts representative features from raw sensory measurements to robustly detect the change of states even in noisy and crowded underground environments. To eliminate the need of a timetable [28, 25] while capturing the temporal characteristics of metro trips, *MetroEye* feeds the statistics of metro running, stopping and individual interchange time, along with the extracted sensor features, into a Conditional Random Field (CRF) model. Compared with the Maximum Entropy Models or HMM adopted in previous works [29, 28], CRF better interprets the temporal connections between the extracted features and passenger's states, as well as the sequential relationships within the three states, which yields more accurate and robust state inference.

We implement *MetroEye* as an Android application, and conduct extensive field evaluations by 32 volunteers in 6 months, covering 14 metro lines of 3 major cities in China (Beijing, Shanghai and Shenzhen). Experimental results demonstrate an overall state identification accuracy of 80.5% at low system overhead, outperforming previous works. In addition to accurately detect passenger states during a metro trip, *MetroEye* further provides two services: (1) an arrival alarm service based on passengers' preset schedule, and (2) aggregating interchange durations from crowdsourced passengers to help passengers plan future routes and guide metro officials for traffic management.

We summarize the key contributions of this paper as follows:

- To the best of our knowledge, *MetroEye* is the first-of-its-kind smartphone-based metro passenger tracking system that continuously and comprehensively monitors the three states (running, stop and interchange) of passengers across multiple metros during an entire trip, which eliminates the need for retyping the next destination when transferring to another metro hastily. *MetroEye* also provides arrival alarms and interchange time statistics to passengers to improve user experience and ease trip planning.

- *MetroEye* combines low power sensors and time statistics of metros into a time sequential model for passenger state inference, which is robust to complex underground noise. The GSM RSSI and time statistic are able to filter in-between stops and eliminate the need of a timetable.

- We evaluate *MetroEye* in a 6-month field study covering 14 major metro lines in 3 cities. Experiment results show *MetroEye* yields competitive performance against previous works and is robust to environmental dynamics.

In the rest of the paper, we review related works in Section 2, followed by an overview and detailed design of *MetroEye* in Section 3 and Section 4. Section 5 presents the system evaluation and we conclude in Section 6.

2. RELATED WORK

MetroEye is related to the following categories of research.

Transportation Mode Estimation. There has been active research on transportation mode detection using smartphones to understand user mobility. Reddy *et al.* [22] utilized GPS and accelerometer to differentiate stationary, walk, biking, and motorized transport. Hemminki *et al.* [13] leveraged accelerometer only to identify more transportation modes including stationary, walk, bus, tram, metro and train. Sankaran *et al.* [23] further harnessed the more power efficient barometer to achieve similar detection accuracies. *MetroEye* is complementary, and focuses on tracking the states of passengers during a metro trip.

Public Transportation Tracking. As public transportation may not strictly accord with a static timetable, researchers have explored smartphones to track public transportation in realtime. EasyTracker [2] aggregated GPS traces to track cars and buses and provided arrival time inference services. Zhou *et al.* [30] leveraged cell tower signals, inertial measurements and audio recordings to track buses and predict arrival time for each bus stop. Transitlabel [8] use mobile phone sensors to recognize the passenger's activities and then labels the interchange semantics. While they can be adopted to metros in principle, the unavailability of GPS as [2] and specific sounds of buses as [30] requires an alternative approach specific to metros. *MetroEye* is inspired by this line of research, and integrates low-power accelerometer, magnetic sensors, cell tower signals, as well as temporal statistics of metros, to track metros and passengers underground.

Metro State Tracking. Yu *et al.* [29] employed acceleration and cell tower signals to estimate the stop and go states of metros. However, the acceleration patterns might be significantly reduced in modern metros [14]. Lee *et al.* [16] exploited ambient magnetic fields in metros to differ their motion states. StationSense [14] further improved the estimation robustness by fusion acceleration and magnetic measurements. However, they cannot well distinguish stops between stations from those at stations, which trigger false arrival reminders. Thiagarajan *et al.* [28] fed acceleration and a fixed timetable into an HMM model to locate metros and is robust to in-between stops. SubwayPS [25] added gyroscope to further track metros between stations. Conversely, *MetroEye* does not rely on a fixed timetable, which can be unavailable in certain countries (*e.g.* China, India). *MetroEye* inputs the statistics of running, stop and interchange time of metros, and the GSM RSSI into a CRF model to get rid of the limitation of static timetable, as well as to deal with in-between stops. By further combining various low-power sensors, *MetroEye* offers more comprehensive metro passenger state tracking (not only stop and go, but also interchange), and implements two services of arrival reminder and interchange time analysis.

3. SYSTEM OVERVIEW

Figure 2 shows the system architecture of *MetroEye*, which consists of two modules, *user tracker* and *service provider*.

User Tracker. This module records data from smartphone sensors including magnetism sensor, accelerometer, GSM and timer. It then extracts distinctive features from each type of sensor data during a metro trip. These retrieved features are integrated by a CRF model. We choose CRF because it interprets the temporal connections between the extracted features and the travelling states, as well as the sequential relationship within the three states. *MetroEye* infers a state by CRF every 20s. The inference interval roughly corresponds to the shortest possible state in the metro trip.

Table 1: Summary of sensory measurements, extracted features, and usage in *MetroEye*.

Sensory Data	Features	Usage
Magnetic Field	Variance	Infer Start of Running State
Acceleration	Energy and period of acceleration difference	Infer States of Running (acceleration, smooth running and deceleration), Stop and Interchange (walking and waiting)
GSM RSSI	Energy, standard deviation, and ratio of energy between first and second halves of time	Filter in-between stops Infer States of Running (launching, smooth running and pulling-in), Stop and Interchange.
Time Statistics	Distributions of running, stop and interchange time	Filter in-between stops Bound sequential state changes during a metro trip

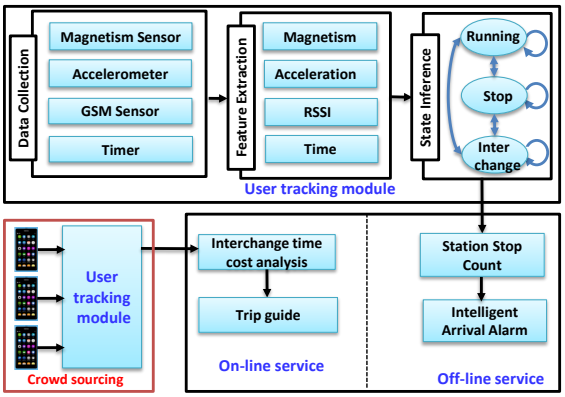

Figure 2: Work flow of *MetroEye*.

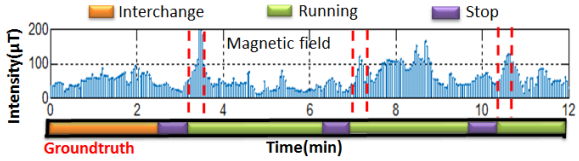

Figure 3: Magnetic field trace of a metro trip.

Service Provider. *MetroEye* provides two services for passengers. (1) *Offline service. MetroEye* delivers intelligent arrival alarms based on the user's preset schedule. Once *MetroEye* detects the next stop is the destination, it vibrates and rings to remind the user. (2) *Online service. MetroEye* crowdsources interchange durations of multiple passengers for interchange time analysis. It helps users to plan routes and guides metro officials for traffic management.

4. SYSTEM DESIGN

This section elaborates on how *MetroEye* extracts robust features from multiple sensory measurements, before feeding into a sequential model for accurate passenger state inference. To robustly infer the three states (*Running*, *Stop* and *Interchange*), *MetroEye* carefully selected sensors to identify different *sub-states*. Table 1 summarizes the sensory measurements used in *MetroEye* and their usage to infer passenger states during a metro trip.

4.1 Magnetic Intensity Sensing

Principle: The contemporary metro operation systems are general driven by the electric engine. This engines, which distribute among each cart, rely on the interaction between winding currents and the magnetic field to generate force. At the acceleration phase, significant alteration of the current triggers notable magnetic field, which generate a considerable torque to push the metro forward [16]. When the metro moves smoothly, the current of motors becomes stable, leading to reduction of magnetic intensity. Thus the magnetic intensity at the beginning of running is normally greater compared to other phases [12].

Measurements: To verify the above phenomenon, we conducted 21 groups of measurements on 5 metro lines in 3 cities. In each experiment, we record magnetic readings in embedded magnetic sensor at a sampling rate of 50Hz. We calculate the magnetic intensity $M(t)$ at time t as:

$$M(t) = \sqrt{(M_x^2(t) + M_y^2(t) + M_z^2(t))} \quad (1)$$

where $M_x(t)$, $M_y(t)$ and $M_z(t)$ represent the magnetic intensity on the X-, Y- and Z-axis at time t, respectively. To filter measurement noise in the raw data, we adopt a moving average window of size 500, which corresponds to a duration of $10s$.

Figure 3 depicts the magnetic field trace of a metro trip. The time series marked blue denotes the variance of magnetic intensity, and the colored bar at the bottom represents the ground truth of three states during the entire trip. As is shown, the magnetic field increases abruptly at the beginning of a running state (marked by the red dashed lines), and keeps at a relatively stable level or descends in other states. Thus an abrupt variance of magnetic intensity can indicate the beginning of the *Running* state.

Magnetic Feature Extraction F_{mag}: We extract magnetic related feature F_{mag} as follows. Given a magnetic intensity time sequence $M = \{M(1,t_1), ..., M(i,t_i), ..., M(N,t_N)\}$, *MetroEye* first filters the raw data using a moving average of size K. Then it searches for the minimal $M(i,t_i)$ and maximal $M(j,t_j)$ within the sequence. If $t_j > t_i$ and the difference $Diff(t) = M(j,t_j)$ - $M(i,t_i)$ exceeds a preset threshold δ_{mag}, it implies a sudden growth of magnetic intensity, and possibly as a result of a metro starts to launch. Conversely, if $Diff(t) < \delta_{mag}$, there might be three possibilities: (1) The metro the passenger takes stops, and the power of the metro's engine approaches zero, which hardly arouses prominent change of magnetic field. (2) The metro is running steadily. In case of little tractive force occurrence, the electric current rarely alters remarkably, and thus the magnetic intensity exhibits stationary. (3) The passenger is walking at an interchange station. In this case, the magnetic field is rarely affected, and remains almost stable. In the work, we extract the magnetic intensity features every $20s$, a general launching time of metros, *i.e.*, $N = 1000$ under the sampling rate of $50Hz$. We empirically set the window size $K = 500$ and the threshold $\delta_{mag} = 85\mu T$ to optimize local experiment performance. *MetroEye* denotes $Diff(t) \geq \delta_{mag}$ as an uprush of magnetic intensity, and regards it as a feature of a metro launching, indicating a *Running* state.

Nevertheless, the above feature has a limitation. As stated above,

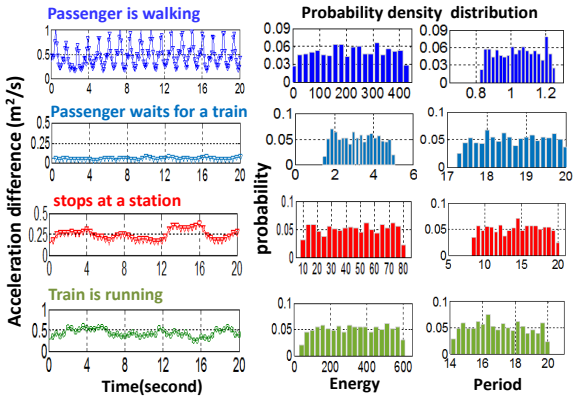

Figure 4: Acceleration traces of four states and the probability density distributions of *energy* and *period*.

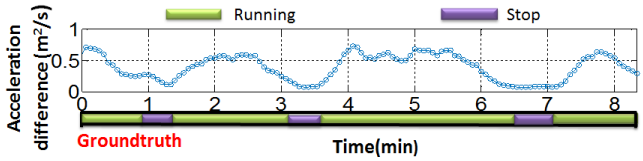

Figure 5: Acceleration trace of a train during a metro trip.

the magnetic intensity is closely relevant to the variance of the electric current. Sometimes, a subway accelerates in a slow pace at the launching phase. The current flow generated by an engine alters mildly, causing moderate variance of the magnetic intensity. Such cases cannot be well realized, which may lead to an inference error. Thus *MetroEye* leverages the variance of the magnetic intensity as only one dimension of features for further inference.

4.2 Acceleration Variance Sensing

Principle: Intuitively, the acceleration variance of a metro keeps rising at the Running phase and drops off when the train is stopping. Such properties have been exploited in previous works for metro tracking [28, 25, 29]. In *MetroEye*, we further extend the use of acceleration to recognize passengers during interchanges, where a passenger either walks towards a platform or waits for a coming metro. Our observation is to detect the periodic walking patterns as well as the relatively stationary state during waiting from acceleration traces for *Interchange* inference.

Measurements. We asked 20 volunteers to record acceleration during their metro trips. The sampling rate is set to $50Hz$ and the volunteer is free to put the smartphone in hand, in pocket or in bags. Given an acceleration trace $A = \{a_1, ..., a_i, ..., a_n\}$, we first calculate the acceleration difference samples as

$$a_i = \sqrt{a_{xi}^2 + a_{yi}^2 + a_{zi}^2} \tag{2}$$

where a_{xi}, a_{yi} and a_{zi} represent the acceleration difference of two successive acceleration amplitudes at each axis as [10]. The difference can well show the variance of acceleration amplitude. We then smooth the acceleration with a window of 50 sample, which approximates the time for a walking step.

Figure 4 displays the acceleration profiles over 20s of four states (passenger walking, passenger waiting, metro running and metro stopping). We define as *energy* $En = \sum_{i=1}^{n} a_i^2$ and *period* as the bin corresponding to the strongest peak by performing Fast Fourier Transformation (FFT) on $\{a_i\}_{i=1}^{n}$. As shown in Figure 4, the

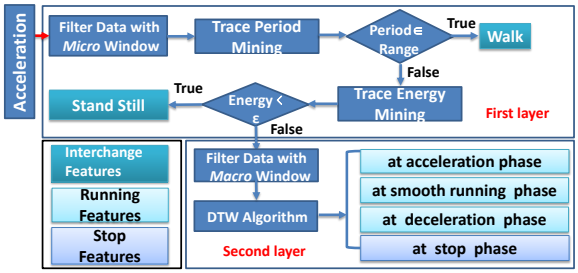

Figure 6: Work flow of acceleration feature extraction.

repetitive pattern of walking exhibits a shortest period within the range of $[0.8s, 1.2s]$, which accords with the walk frequency of human being spans from $0.5Hz$ to $2Hz$ [20]. Such a *period* (accord with walking frequency) is hardly observed in acceleration traces for non-walking states. We also observe that the *energy* of the acceleration is relatively low when a passenger is waiting for a metro, while slightly larger ($10 \sim 80$) when the passenger is on a halted metro. Such a difference in *energy* even if the passenger is stationary comes from the vibration of the metro's running engine, which cannot be smoothed by a window of 50 samples. The energy for a walking passenger and a running metro is often dramatically larger, due to the larger acceleration amplitudes. Inspired by the above observations, we choose *energy* and *period* to identify *Interchange* states (passenger walking towards or waiting at platforms).

To further investigate the acceleration characteristics of metro, we increase the observation interval to 8 minutes, which can cover an entire process of metro acceleration, steadily running, deceleration and stopping. We also increase the rolling average window to 1000 samples accordingly. Figure 5 plots the acceleration traces of a train during a metro trip. The blue curve indicates the acceleration difference, and the bar at the bottom denotes the ground truth of two marked states. As expected, the acceleration difference climbs up remarkably until the subway runs smoothly. When the train approaches the next station, the acceleration difference decreases gradually to a low level. Therefore we include both short-term and long-term features from acceleration for state inference.

Acceleration Feature Extraction F_{acc}: As shown in Figure 6, the work flow of acceleration feature extraction consists of two layers. The first layer analyzes the short-time acceleration patterns of *Interchange*, including individual *walking* and *waiting*. The second layer learns long-term acceleration profiles of *Running* and *Stop*, which are further split into *acceleration*, *smooth running*, *deceleration* and *stop*.

In the first layer, after collecting a 1000-sample acceleration sequence within an *Inference interval*, *MetroEye* first computes its difference, and smooths it with a micro rolling average window of 50 samples. Then *MetroEye* calculates trace period and judges whether it lies in the range $[0.5s, 2s]$. If yes, *MetroEye* confirms it as *walking*. Otherwise, *MetroEye* further calculates its energy. If the energy is lower than a predefined threshold $\epsilon = 5$, *MetroEye* announces it as *waiting*. If not, *MetroEye* considers it as a metro trace, and leaves the difference trace to the second layer.

In the second layer, *MetroEye* adopts a Dynamic Time Warping (DTW) [1] algorithm to recognize the metro's acceleration profiles (*acceleration*, *smooth running*, *deceleration* and *stop*). We choose DTW because the acceleration measured on a metro is susceptible to environmental dynamics *e.g.* the passenger's body movement, which makes simple peak detection [4] or threshold crossing counting [3] erroneous. Conversely, DTW is suitable to measure similarity between two temporal sequences that vary in time or speed. Given an acceleration difference trace from the first layer, *Metro-*

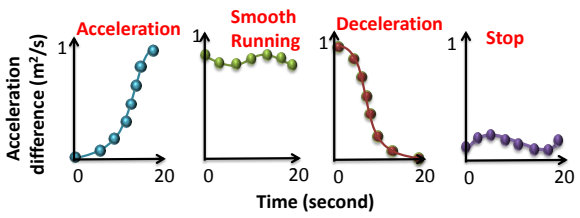

Figure 7: The diagram of the templates in DTW.

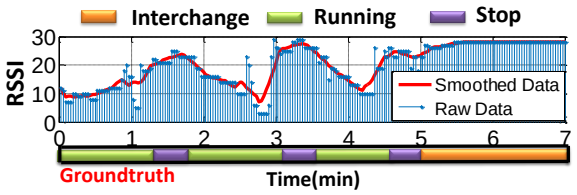

Figure 8: RSSI trends in a metro trip.

Eye computes 1000-sample windowed means (macro smoothing window), corresponding to the length of acceleration trace. Afterwards, *MetroEye* compares the smooth acceleration sequence with the pre-stored templates listed in Figure 7, which are generated by the training set, and then calculates their DTW distances. Finally, the testing trace is classified as the one with the minimal distance. *MetroEye* takes the corresponding result as a feature to infer the passenger's state. Note, however, that it is difficult to differentiate stops at stations and those between station [28]. Therefore *MetroEye* incorporates with cell tower signal strength (*i.e.* GSM RSSI) and time statistics to filter out in-between stops, which will be discussed later.

4.3 GSM Signal Sensing

Principle: GSM is the main communication medium for a metro. As GSM signals can be attenuated during a metro trip, the variance of GSM received signal strength index (RSSI) may indicate different states during a metro trip. Generally, a metro station is installed with several child cell-sites to guarantee the communication quality underground, while few cell-sites are installed in the metro tunnels, leading to the weaker signal strength. Thus RSSI holds potential to filter in-between metro stops in the tunnels.

Measurements: Figure 8 illustrates a trace of GSM RSSI during a metro trip. The blue bar plots the raw RSSI, and the red line illustrates the RSSI after smoothing. As is shown, the RSSI descends gradually as the metro departs from the station, and keeps at a low level in the middle of tunnel. When the train approaches the next station, the RSSI climbs up and arrives at a peak while the metro halts at a station. In addition, when passengers are transferring in an interchange station, the RSSI retains high. In our experiment, we parse the RSSI trace every 20s, and classify it as one of the five RSSI diagrams in Figure 9. The first three diagrams denote the distinct RSSI trends of *Running* state when passengers are on metros at *launching*, *smooth running* and *pulling-in* phases. We define *launching* as the $20s$ after a metro departs from a station, *pulling-in* as the $20s$ before a metro halts at a station, and *smooth running* as the rest during metro *Running*. The fourth and fifth diagrams plot the RSSI profiles of *Stop* and *Interchange*, respectively. As is shown, the RSSI profiles for each class exhibit a unique pattern. In this measurement, we have collected 270 segments of RSSIs for each class, lasting around 1.5 hours.

RSSI Feature Extraction F_{rss}: Given an RSSI sequence $R = \{r_1, r_2, ...r_n\}$ smoothed by a rolling window, we adopt three efficient criteria to capture the distinctive trends as shown in Figure 9. The first is energy $En = \Sigma_{i=1}^{n} r_i$, where r_i denotes the strength of ith sample in the sequence. The second is the ratio of RSSI energies in the first half of time to the last half of time. It is calculated as $rfl = \frac{En(f)}{En(l)}$, where $En(f) = \Sigma_{i=1}^{n/2} r_i$ and $En(l) = \Sigma_{i=n/2}^{n} r_i$. It indicates the transition of RSSI energy of the sequence. For instance, *rfl* at *launching* is larger than that at *pulling-in* in Figure 9. The last is the standard derivation *std*. The RSSI of *Interchange* usually exhibits small *std*.

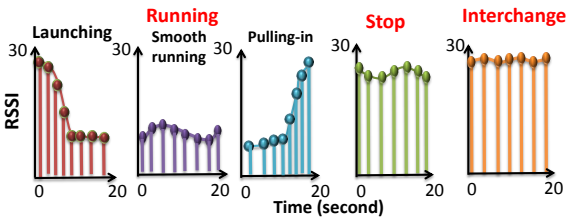

Figure 9: Diagram of GSM RSSI profiles.

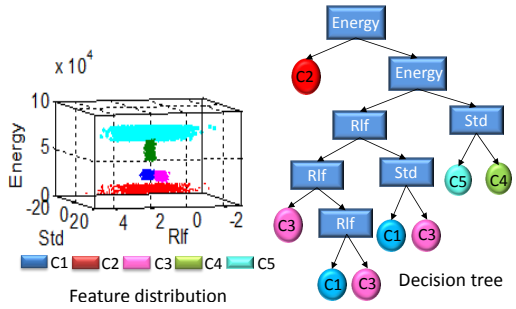

Figure 10: RSSI feature distribution and its decision tree ($C1$ to $C5$ denotes the five RSSI profiles in Figure 9).

The left of Figure 10 illustrates the distribution of the three criteria. As is shown, the five RSSI profile classes are divided into five clusters by the three criteria, and hence can be classified by a decision tree model as on the right of Figure 10. Specifically, the paths from root to leaf represent classification rules, and the splitting features and thresholds are determined depending on the information gain calculated by entropy. *MetroEye* adopts the above decision tree to recognize each phase of metro from the GSM RSSI profiles, and leverages it to infer the passenger's trip status.

4.4 Time Statistics

Principle: Even though metro timetables are naturally modified based on the actual requirement such as the boarding crowd size, the running time between two successive stations, the stop time at a station and the time cost at an interchange usually alter within a certain region. *MetroEye* leverages the statistics of running, stop, and interchange times to filter in-between stops and bound the sequential state changes during a metro trip.

Temporal Feature Extraction F_{tim}: To acknowledge the temporal characteristics of the travelling process comprehensively, we conducted 27 groups of motivating experiments across 9 metro lines in 3 cities during both peak and off-peak hours and for both weekdays and weekends. Three criteria are considered: the metro's running time between two consecutive stations, the halt time at a station, and the individual time cost in the interchanges, whose distributions are illustrated in Figure11. According to our measurements, more than 90% of the running time aggregates within

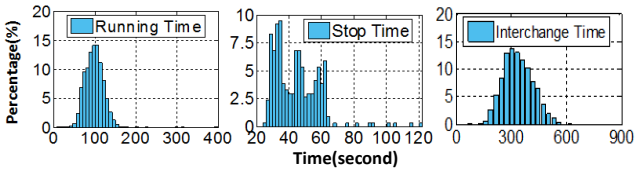

Figure 11: Time distribution in the metro trip

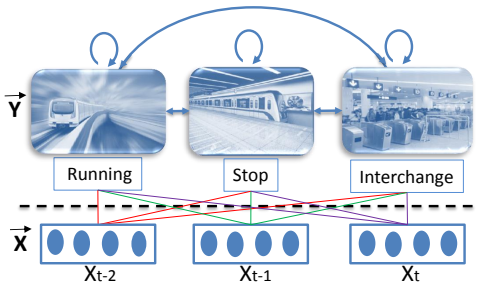

Figure 12: Diagram of the linear-chain CRF.

$[60s, 180s]$. Around 80% of the stop time lies within $[25s, 60s]$. And 95% of the interchange time is less than $10min$. Accordingly, the three criteria enable to depict the temporal relationships of the inference stages. For example, the predicted running time is supposed to corroborate the distribution of the actual running time, and the inferred train's halt interval also should be in accord with the range of the actual stop time. Therefore, *MetroEye* records the three time parameters by the smartphone timer to model the inner temporal connections of the predicted states.

4.5 Inference Model

We adopt a classic graphical model, conditional random field (CRF) [15], to depict the temporal relations between the observable features as well as the connections in the hidden stages. CRFs are discriminative probabilistic graphical models for analyzing the sequential data, which have been applied to numerous real sequential issues such as natural language processing [15], biological gene sequencing [7] and stage tracking [11, 9]. It calculates a single log-linear distribution over label sequences $\overrightarrow{Y}$ based on particular observation features $\overrightarrow{X}$ by maximizing the conditional probability $p(\overrightarrow{Y}|\overrightarrow{X})$.

Compared to HMMs, CRFs allow features to depend on several states to account for long-term effects. Moreover, it also outperforms the maximum entropy markov models [19] by resolving the label bias issue [15]. These advantages enable CRF to characterize the relations between the observable contextual features and the hidden states in more depth, making it more suitable for passenger tracking underground.

Figure 12 shows the diagram of a linear-chain CRF employed in *MetroEye*. Specifically, $\overrightarrow{Y} = \{Y_1, Y_2, \ldots, Y_t\}$ indicates the hidden variable sequence, where $Y_t \in \{Interchange, Running, Stop\}$ represents the predicted state of *inference interval* t. Given that the state might change during an *inference interval*, the state is labeled as the one with the longest period. For instance, if a metro runs into a station in the first $5s$, and halts at the station for the last $15s$, the state is labeled as *Stop*. The observation sequence is given by $\overrightarrow{X} = \{X_1, X_2, \ldots, X_t\}$, where $X_t \in \{F_{mag}, F_{acc}, F_{rss}, F_{tim}\}$ are the corresponding features extracted by *MetroEye* at *Inference interval* t. For a label sequence $\overrightarrow{Y}$ with $N+1$ variables, its conditional probability $p(\overrightarrow{Y}|\overrightarrow{X})$ on the entire observation sequence $\overrightarrow{X}$ is given by the normalized form of potential functions,

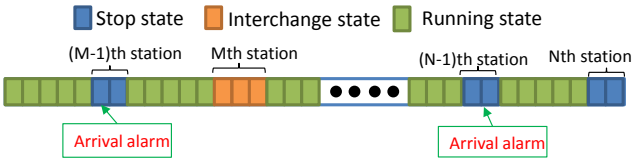

Figure 13: A diagram of an arrival alarm.

$$p(\overrightarrow{Y}|\overrightarrow{X}) = \frac{1}{Z_{\overrightarrow{\lambda}}(\overrightarrow{X})} exp\Big(\sum_{t=1}^{N}\sum_{i=1}^{M}\lambda_i f_i(Y_{t-1}, Y_t, \overrightarrow{X}, t)\Big), \quad (3)$$

where $Z_{\overrightarrow{\lambda}}(\overrightarrow{X})$ is the normalization factor, $f(Y_{t-1}, Y_t, \overrightarrow{X}, t)$ is a real-valued feature function to express the empirical distribution of the training data, and λ is the weight of each feature function [15]. Specifically, the M feature functions consist of two parts,

$$\begin{aligned}&\sum_{i=1}^{M}\lambda_i f_i(Y_{t-1}, Y_t, \overrightarrow{X})\\ =\ &\sum_{k=1}^{m}\nu_k fs_k(Y_t, \overrightarrow{X}, t) + \sum_{j=1}^{n}\mu_j ft_j(Y_{t-1}, Y_t, \overrightarrow{X}, t) \quad (4)\end{aligned}$$

The first part is the state feature function $fs(Y_t, \overrightarrow{X}, t)$, which describes the correlations between the observation feature sequence $\overrightarrow{X}$ and the label at the current phase t. The second part is the transition state function $ft(Y_{t-1}, Y_t, \overrightarrow{X}, t)$, establishing the transition relations of the observation sequence $\overrightarrow{X}$ with the previous and current hidden states at the intervals $t-1$ and t. This function can well characterize the temporal associations of the contextual features such as the acceleration and RSSI with their former and current states. The weights ν and μ of these feature functions are calculated by maximizing the conditional log-likelihood of the labelled sequences.

We adopt the L-BFGS [17] for an efficient training, and apply the Viterbi Algorithm [18] to infer the hidden states based on the potential functions.

4.6 Service Provider

MetroEye provides two services based on the inferred *Stop*, *Running*, and *Interchange* states. (1) *Arrival Alarm*, an off-line service to remind users to get off the metro in time. (2) *Interchange Time Analysis*, which calculates the statistics of per-station interchange time to suggest passengers of a more efficient trip and guide metro officials to devise reasonable construction / retrofitting plans. This subsection presents the work flows of the two services, and we defer the detailed performance evaluation to Section 5.5.

4.6.1 Arrival Alarm

For arrival alarms, *MetroEye* merely requires the user to log how many stops to the terminal and interchanges from his/her origin, releasing the burden of reloading maps once the subway line changed. Unlike previous works [29, 14] which require a user to input trip information for each metro, *MetroEye* avoids retyping such information when hastily transferring to another metro and significantly improves user experience. Based on the trip information, *MetroEye* infers whether the metro has reached a station by detecting the *Stop* states (note *MetroEye* is able to filter stops between stations), and counts the number of stations the metro has passed.

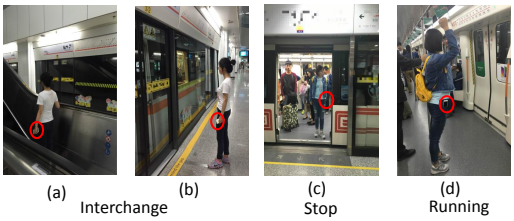

Figure 14: Illustration of experiment cases.

Table 2: Summary of experimental dataset.

Number / City / State	Beijing	Shanghai	Shenzhen
In an interchange	23500	38513	30487
On a stop train	25445	48800	33255
On a running train	44403	63055	44503

In the default setting, *MetroEye* reminds the user one station ahead of the destination. For instance, if a user indicates that the Nth station is the terminal and the Mth station is an interchange, then *MetroEye* reminds the user at the $(M-1)th$ and $(N-1)th$ station. Figure 13 shows a diagram of the arrival alarm service.

4.6.2 *Interchange Time Analysis*

Unique in *MetroEye*, the *Interchange* states of passengers are explicitly inferred during a metro trip. For interchange time analysis, a user is asked to input the name of the interchange station. Then *MetroEye* automatically records the time after detecting an *Interchange* state. By collecting individual interchange durations from multiple users and calculating the average time spent at each interchange station at the server end, *MetroEye* provides suggestions for passengers to plan future trips. Such statistics of interchange time can also benefit metro designers to optimize metro schedules during rush hours.

5. EVALUATION

This section presents the evaluation methodologies and detailed performance of *MetroEye*.

5.1 Experimental Setup

We implement *MetroEye* as an Android application. Each participant installs *MetroEye* on their smartphone and launches it at the beginning of each metro trip. Afterwards, *MetroEye* invokes the corresponding sensors to collect measurements. The participant manually labels the states of the sensory measurements for each *inference interval* based on his/her route. If more than one state occur in the *inference interval*, we label the interval as the majority state. This case represents less than 3% of our dataset. We use these labels as ground truth in our evaluations. In total, 3840 sets of metro trips have been conducted by 32 volunteers within 6 months, covering 14 metro lines of Beijing, Shanghai and Shenzhen, where each metro trip lasts for 30 minutes on average. During the measurements, the volunteers are free to stand or sit in a cart, or exchange at interchanges, and put their smartphone in hand, in pockets or in bags (Figure 14). Table 2 summarizes our dataset of labelled states. We randomly retrieve 80% of the dataset for training, and the rest 20% for testing.

We also implement two representative metro tracking schemes [29, 14] as baselines. Participants are asked to record the states inferred by these two approaches as well during the data collection.

5.2 Micro-benchmarks

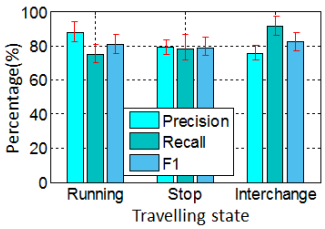

Figure 15: 10-fold cross validation of CRF.

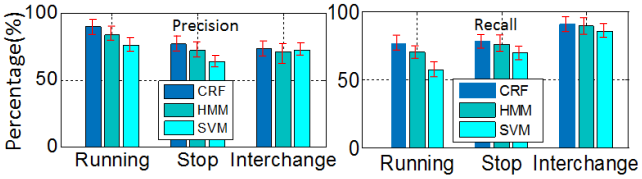

Figure 16: Comparison of inference model.

Table 3: Effectiveness of features.

	On a running train		On a stop train		In an interchange	
Feature	Precision	Recall	Precision	Recall	Precision	Recall
F_{mag}	14.1%	21.8%	9.3%	8.1%	3.7%	5.2%
$F_{mag}+F_{acc}$	31.7%	39.7%	27.3%	33.5%	59.9%	51.7%
$F_{mag}+F_{acc}+F_{rss}$	70.3%	69.4%	62.9%	64.3%	72.1%	88.6%
$F_{mag}+F_{acc}+F_{rss}+F_{tim}$	89.6%	76.2%	77.2%	78.3%	73.4%	90.2%

5.2.1 *Effectiveness of CRF model*

We first conduct a 10-fold cross validation to evaluate the performance of CRF model. As shown in Figure 15, the F1-measures of *Running* and *Interchange* states surpass 80%. Although the F1-measure of *Stop* state is slightly lower, it still maintains at a reasonable value of 78.7%. Furthermore, the standard deviation of each state illustrated by error bars is less than 10%, which demonstrates the competitive and robust inference performance of CRF model.

We then compare the inference performance of CRF with other two classic models: (1) HMM, a statistical model known for its applications in temporal pattern recognition; (2) SVM, which can efficiently perform non-linear classification by using the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Figure 16 shows the comparison of model performances. CRF outperforms HMM and SVM in terms of the average precision and recall, which verifies the advantages of adopting CRF model.

5.2.2 *Effectiveness of Features*

To show the effectiveness of the selected features, we observe the variance of inference performance of the CRF model by adding features when training it. The results are shown in Table 3. As can be seen, the precision and recall of each state are improved with the growth of features, indicating that the selected features are helpful for prediction.

5.3 System Accuracy

5.3.1 *Inference Accuracy*

The confusion matrix in Figure 17 shows the confusion matrix of *MetroEye* to infer the three states. Each column is the ground truth, and each row denotes the corresponding inference of *MetroEye*. The data listed in the middle is the number of *inference interval* of each state. We can see that the recalls of three states are all above 75%, with a peak value at 90.2%. Likewise, the precisions of *Running* and *Stop* are also over 75%. The precision of *Interchange*

Ground truth	Predictions			
	Running	Stop	Interchange	
Running	**23155**	3744	3493	76.2% (Recall)
Stop	2119	**16839**	2541	78.3% (Recall)
Interchange	577	1232	**16691**	90.2% (Recall)
	89.6%	77.2%	73.4%	80.5% (Accuracy)
	Precision			

Figure 17: Inference accuracy of *MetroEye*.

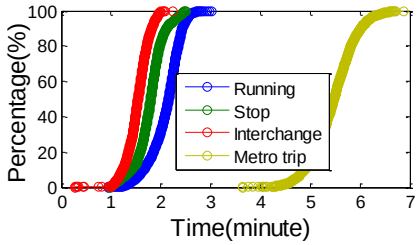

Figure 18: Accumulate error time.

System \ State	Running		Stop	
	Precision	Recall	Precision	Recall
MetroEye(2 states)	91.0%	84.5%	83.7%	87.2%
MetroEye(3 states)	89.6%	76.2%	77.2%	78.3%
Baseline1	74.2%	71.3%	68.1%	74.6%
Baseline2	75.3%	74.1%	65.6%	70.2%

Figure 19: Performance comparison with baselines.

state is lower. This is mainly due to the relatively short interchange time during a metro trip, and the limited number of samples for *Interchange* states. However, the precision of *Interchange* still remains at an acceptable level of 73.4%, and its F1-measurement is over 80%. Figure 18 further examines the timing error during each metro trip. 97% of the timing errors for *Interchange*, *Stop* and *Running* are about 2 minutes, 2.3 minutes, and 2.7 minutes, respectively. The accumulate time error of overall metro trip, therefore, is under 7 minutes. Although the inference performance of *Interchange* state is not as good as that of *Running* state, its timing error is much shorter, due to its limited proportion of a metro trip.

By comparing the confusion matrix (Figure 17), we find the inference errors are mainly caused by two reasons: (1) The *Interchange* state that a passenger is waiting for the next metro tends to be easily mistaken as a *Stop* state, and vice versa due to the sometimes similar ambient characteristics of the two states. (2) Most errors occur in the transition periods of two states. This is because the contextual features may not immediately change when a new state starts, and *MetroEye* may fail to detect such transitions. However, *MetroEye* still achieves an overall accuracy of 80.5%. We plan to enhance the detection of state changes to improve the overall accuracy in the future work.

5.3.2 *Comparison with Baselines*

We compare the inference accuracy of *MetroEye* with two representative metro tracking schemes in [14] (Baseline 1) and [29] (Baseline 2). The two baselines are able to detect *Running* and *Stop* states of a metro independent on the fixed timetable, which have the similar working principles of *MetroEye*. We compare the inference accuracy of *MetroEye* with the two baselines in two ways. (1) We modify *MetroEye* to detect two states only and ignore the

Brand	CPU	RAM	ROM	Power Capacity	Operation System
Galaxy S6	8-cores 2.1GHz	3GB	32GB	2550mAh	Android 5.0
HTC Desire A6	8-cores 1.7GHz	2GB	16GB	2600m Ah	Android 5.0
HUAWEI 4C	8-cores 1.2GHz	2GB	8GB	3100mAh	Android 4.4
LenovoK80M	4-cores 1.8GHz	4GB	64GB	4000mAh	Android 4.4

Figure 20: Configurations for system overhead evaluation.

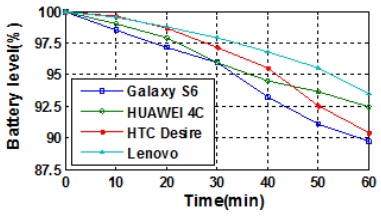

Figure 21: Power consumption.

Interchange state. In this case, metro trips with interchanges are divided into several trips by the interchanges. And we re-train the CRF model to restrain to two states. (2) Directly compare *MetroEye* with the two baselines. Figure 19 illustrates the performance comparison of *MetroEye* and the baselines. We can see the modified *MetroEye* that detects two states performs the best. Moreover, even though the unmodified *MetroEye* detects one more state, its inference accuracy still outperforms the two baselines.

By carefully checking the inference errors, we find that most errors of Baseine 1 come from mistaking the spurious stops in the tunnel as stops at stations. This is because Baseline 1 only adopts magnetic field and acceleration for state inference, which can hardly filter in-between stops by analyzing the magnetic variance and acceleration pattern. For the approaches of Baseline2, its main errors come from the noisy acceleration measurements. The user's body movement or the dynamic jolts of trains makes the simple threshold-based methods erroneous [14]. Differently, *MetroEye* integrates multiple effective sensory data by a representative sequential model to describe the user's trip, thus dramatically improving the robustness of passenger state inference even in noisy and dynamic environments without the assistance of fixed timetable.

5.4 System Overhead

Since *MetroEye* targets at continuous metro passenger tracking, it is important to evaluate its overhead on the energy-constrained smartphones. This subsection presents the power consumption, CPU utilization and system delay of *MetroEye* using four popular commercial smartphones. Figure 20 summarizes the configurations of the smartphones used for system overhead evaluation.

5.4.1 *Power consumption*

We install *MetroEye* on the above smartphones and download a logger [21] to monitor the battery usage in 5 metro trips. Each trip lasts more than one hour. Figure 21 illustrates the average battery level as a function of the execution time of *MetroEye*. As is shown, *MetroEye* consumes negligible power (less than 2%) every 10 minutes. The battery level follows a gradual downward trend as time goes by, and ends in 89.5% for Galaxy S6, 92.4% for HUAWEI 4C, 90.8% for HTC Desire, and 93.2% for Lenovo after 1 hour. The results indicate that *MetroEye* is power efficient and is feasible for continuous metro trip tracking.

5.4.2 *CPU utilization*

To measure the CPU usage of *MetroEye*, we install an application [5] on each smartphone to monitor the CPU occupation during 5 metro trips. The results are shown in Figure 22, which indicates

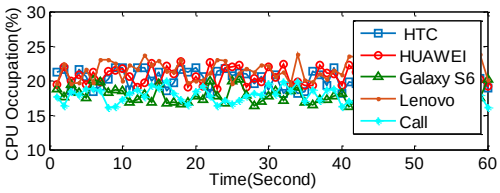

Figure 22: CPU utilization.

State / Phones	Feature Extraction				Inference Model	Total Time
	Magnetic Intensity	Acceleration	RSSI	Time		
Galaxy	0.04s	0.19s	0.11s	0.01s	0.31s	0.5s
HTC	0.05s	0.19s	0.14s	0.01s	0.32s	0.51s
HUAWEI	0.08s	0.20s	0.14s	0.01s	0.31s	0.51s
Lenovo	0.08s	0.22s	0.13s	0.01s	0.32s	0.54s

Figure 23: System delay.

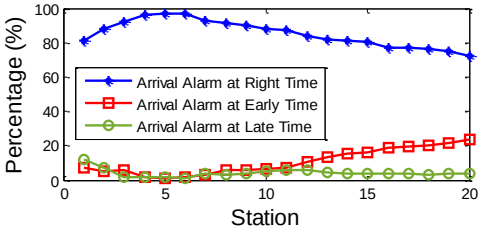

Figure 24: Performance of arrival alarm.

that the average CPU utilizations of the four smartphones are kept at a relatively low and stable level, with a little fluctuation centered around 20%. Compared with the CPU usage of around 15% to 22% for a phone call with the 8-cores CPU, which is marked by the cyan colored line, the occupation of *MetroEye* is acceptable in daily use.

5.4.3 System delay

The delay of *MetroEye* comes from two tasks, namely, feature extraction and model inference. We launch a time logger to record the durations of each component. The results are shown in Figure 23. As *MetroEye* invokes multi-treads to process sensory data in parallel, the total time is the sum of the maximum processing time among the sensors for feature extraction and the time of model inference. As illustrated in Figure 23, the longest processing time among the feature extraction is due to the acceleration module, fluctuating around 0.2s, which is mainly caused by the multiple index calculation and Dynamic Time Warping algorithm. Additionally, the inference time of CRF varies around 0.3s. Therefore, the final system delay adds up to around 0.5s. Compared to the *inference interval* of 20s, this delay has negligible effect on system performance.

5.5 Case Studies

In this section, we show two case studies to demonstrate the services *MetroEye* can provide.

5.5.1 Arrival Alarm

In the first, we investigate the relationship between the accuracy of arrival alarms and the trip length. The trip length is measured by the number of stations the metro has passed.

The blue curve in Figure 24 indicates the accuracy of arrival alarm initiated at one station ahead. As is shown, the accuracy of arrival alarm keeps increasing for the first 4 to 5 stations, and descends gradually as the trip becomes longer, finally reaching at around 70%. This is because *MetroEye* has not collected sufficient contextual data at the beginning. After 3 or 4 stations, *MetroEye* gradually collects enough measurements to build the temporal relationships of observable features and hidden travelling states. Thus the accuracy improves. With time passing by, the inference error continuously accumulates, and eventually deteriorates the alarm performance. Nevertheless, the alarm performance still maintains about 70% for a 20-station trip.

We also take an analysis of the errors of arrival alarms. The red and green curves in Figure 24 demonstrate the errors of early and late arrival alarms, respectively. As is shown, the percentages of both kinds of errors remain low during the first 10 stops. As the number of stops increases, the occurrence of late arrival alarms still keeps low, while that of early arrival alarms steadily grows. However, about 80% of the early arrival alarms are at most two stops earlier than the expected destination. Based on the above error analysis, we may conclude that *MetroEye* accurately reminds the passengers to get off with a high accuracy. It tends to remind the passengers ahead of his/her destination for long metro trips, but seldom misses the right destination (*i.e.*, late alarms) for both short and long metro trips.

5.5.2 Interchange Time Analysis

We evaluate the service of interchange time analysis by figuring out the stations with long-term interchange time in three major cities in China.

According to the experimental results in Section 4.4, more than 95% interchange durations are shorter than 600s. Hence, we separate short interchange time from long interchange by 600s. Since *MetroEye* predicts a state every 20s, we calculate the interchange time by merging the durations of the consecutive *Interchange* states. For example, if *Interchange* states are detected in three continuous *inference intervals*, the interchange time is recorded as 60s. We use the above method to count the long and short interchange time inferred by *MetroEye* in the testing dataset. Figure 25 illustrates the ratio of interchange times in two categories. We can see 7% of interchange times are over 600s. Based on the interchange names provided by users, we also mark their corresponding locations with red nodes in Figure 26. As is shown, the problem of long-term transfer exists in 11 interchange stations. The main reasons for a long interchange time include overcrowded passage and improper station structure, *e.g.*, limited stairs and elevators. Accordingly, metro officers can control the crowd timely, or recommend the metro designer to improve the infrastructure.

6. CONCLUSION

Designing metro trip tracking systems for the passenger is critical yet challenging. Existing solutions focus on the mobility of a single metro rather than the passenger's state during an entire trip (may consist of multiple subway lines). In this paper, we propose *MetroEye*, a passenger's metro trip tracking system based on smartphones, which integrates underground context signals with a CRF model to infer the three states defined as *Running, Stop, Interchange* in a metro trip. *MetroEye* carefully characterizes the time sequence of the three states, and continuously monitors passenger states even during metro exchanges, which waives the requirement of user's multiple inputs during the trip. The performance of *MetroEye*, evaluated on a dataset covering 14 metro lines in 3 major cities within 6 months, is promising. Its overall system accuracy is up to 80.5%, outperforming the state-of-the-arts.

7. REFERENCES

[1] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proc. ACM KDD Workshop*,

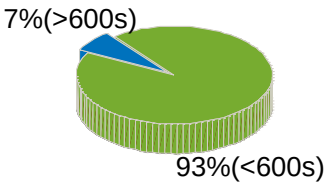

Figure 25: Distribution of interchange time.

Figure 26: Stations with interchange time over 600s.

pages 359–370, 1994.

[2] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson. Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In *Proc. ACM SenSys*, pages 68–81, 2011.

[3] A. Brajdic and R. Harle. Walk detection and step counting on unconstrained smartphones. In *Proc. ACM UbiComp*, pages 225–234, 2013.

[4] J. Chon and H. Cha. Lifemap: A smartphone-based context provider for location-based services. *Pervasive Computing*, (2):58–67, 2011.

[5] CPU Monitor. https://play.google.com/store/apps/details?id=com.glgjing.stark.

[6] Subway delays skyrocket because riders lack basic manners. http://nypost.com/2015/02/24/overcrowding-on-subways-leads-to-massive-spike-in-delays/.

[7] D. DeCaprio, J. P. Vinson, M. D. Pearson, P. Montgomery, M. Doherty, and J. E. Galagan. Conrad: gene prediction using conditional random fields. *Genome Research*, 17(9):1389–1398, 2007.

[8] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino. Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 193–206. ACM, 2016.

[9] W. Gu, L. Shangguan, Z. Yang, and Y. Liu. Sleep hunter: Towards fine grained sleep stage tracking with smartphones. *IEEE Transactions on Mobile Computing*, 15(6):1514–1527, 2016.

[10] W. Gu, Z. Yang, L. Shangguan, X. Ji, and Y. Zhao. Toauth: Towards automatic near field authentication for smartphones. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 229–236. IEEE, 2014.

[11] W. Gu, Z. Yang, L. Shangguan, W. Sun, K. Jin, and Y. Liu. Intelligent sleep stage mining service with smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 649–660. ACM, 2014.

[12] M. HAVAS, S. SHUM, and D. Raheman. Passenger exposure to magnetic fields on go-trains and on buses, streetcars, and subways run by the toronto transit commission, toronto, canada.

[13] S. Hemminki, P. Nurmi, and S. Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proc. ACM SenSys*, pages 1–14, 2013.

[14] T. Higuchi, H. Yamaguchi, and T. Higashino. Tracking motion context of railway passengers by fusion of low-power sensors in mobile devices. In *Proc. ACM ISWC*, pages 163–170, 2015.

[15] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ACM ICML*, pages 282–289, 2001.

[16] G. Lee and D. Han. Subway train stop detection using magnetometer sensing data. In *Proc. IPIN*, pages 766–769, 2014.

[17] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[18] H. L. Lou. Implementing the viterbi algorithm. *Signal Processing Magazine*, 12(5):42–52, 1995.

[19] A. McCallum, D. Freitag, and F. C. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proc. ACM ICML*, pages 591–598, 2000.

[20] A. Pachi and T. Ji. Frequency and velocity of people walking. *Structural Engineer*, 83(3), 2005.

[21] PowerTutor. https://play.google.com/store/apps/details?id=edu.umich.PowerTutor.

[22] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using mobile phones to determine transportation modes. *Transactions on Sensor Networks*, 6(2):13:1–13:27, 2010.

[23] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh. Using mobile phone barometer for low-power transportation context detection. In *Proc. ACM SenSys*, pages 191–205, 2014.

[24] Shanghai Metro. https://en.wikipedia.org/wiki/Shanghai_Metro#Incidents.

[25] T. Stockx, B. Hecht, and J. Schöning. Subwayps: Towards smartphone positioning in underground public transportation systems. In *Proc. ACM SIGSPATIAL*, pages 93–102, 2014.

[26] Introduction to Subway Ridership. http://web.mta.info/nyct/facts/ridership/.

[27] List of New York City Subway services. https://en.wikipedia.org/wiki/List_of_New_York_City_Subway_services.

[28] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In *Proc. ACM SenSys*, pages 85–98, 2010.

[29] K. Yu, H. Zhu, H. Cao, B. Zhang, E. Chen, J. Tian, and J. Rao. Learning to detect subway arrivals for passengers on a train. *Frontiers of Computer Science*, 8(2):316–329, 2014.

[30] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. ACM MobiSys*, pages 379–392, 2012.