8-2017

Smartphone sensing meets transport data: A collaborative framework for transportation service analytics

Yu LU
Beijing Normal University

Archan MISRA
Singapore Management University, archanm@smu.edu.sg

Wen SUN
Xidian University

Huayu WU
Institute for InfoComm Research

Published in IEEE Transactions on Mobile Computing, 2017 August
http://doi.org/10.1109/TMC.2017.2743176

Smartphone Sensing Meets Transport Data: A Collaborative Framework for Transportation Service Analytics

Yu Lu, Archan Misra, Wen Sun, and Huayu Wu,

Abstract—We advocate for and introduce *TRANSense*, a framework for urban transportation service analytics that combines participatory smartphone sensing data with city-scale transportation-related transactional data (taxis, trains etc.). Our work is driven by the observed limitations of using each data type in isolation: (a) commonly-used anonymous city-scale datasets (such as taxi bookings and GPS trajectories) provide insights into the aggregate behavior of transport infrastructure, but fail to reveal individual-specific transport experiences (e.g., wait times in taxi queues); while (b) mobile sensing data can capture individual-specific commuting-related activities, but suffers from accuracy and energy overhead challenges due to usage artefacts and lack of appropriate sensing triggers. *TRANSense* demonstrates how a judicious fusion of such disparate data sources can overcome these challenges and offer novel insights. We detail two examples: (a) Taxi Service Analyzer that provides accurate detection of commuter queuing for taxis and estimates their wait time, by using taxi trip records to identify potential taxi locations with high demand and subsequently selectively triggering mobile sensing-based queuing analytics on nearby commuters; and (b) Subway Boarding Analyzer that identifies instances when passengers fail to board arriving trains, by first estimating train arrivals from temporal patterns of passenger egress at station gantries, and then using mobile sensing-based analysis of commuter movement behavior on platforms. Experiments with real-world datasets (from over 20,000 taxis and 1.7 million commuters in Singapore) show the power of this approach: the taxi service analyzer detects commuter queuing with over 90% accuracy with negligible energy overhead and estimates wait times with error margins below 15%, whereas the subway boarding analyzer can detect failed boarding events with a precision of over 90% (more than thrice what is achievable through purely mobile sensing).

Index Terms—Data integration, public transportation, data analysis, crowdsourcing, pervasive computing.

1 Introduction

Developing adaptive and personalized public transportation services is a key component of future smart city initiatives. To support such adaptive services, transportation analytics research presently adopts one of two distinct approaches: (a) The *infrastructure-driven* approach utilizes transactional informatics data that are increasingly becoming available from public transportation information systems (e.g., taxi trajectories and logs [1], smart card usage history for subway or bus rides [2]); while (b) The *participatory sensing* approach [3] utilizes data from smartphone-embedded inertial & location sensors (e.g., GPS, accelerometer and compass), obtained from a pool of participating commuters.

Both these approaches have their own merits and demerits: the infrastructure-based approach is usually comprehensive and accurate (it typically has visibility on the movement history and trip history of *every* vehicle) for understanding and predicting aggregate traffic characteristics and patterns, but cannot observe an individual commuter's "personal commuting experience". For example, it cannot reveal how long a person had to wait at a taxi stand prior to boarding. In contrast, participatory mobile sensing can capture an individual's commuting experience, but clearly provides only a (possibly non-representative) sampling of the overall state of the transportation infrastructure. For example, it cannot reveal how many people board or disembark from a subway train at the stations along its route. Moreover, continuous mobile sensing-based recognition of commuting activities can suffer from both *accuracy degradation* (inertial sensors are notoriously sensitive to usage-driven artefacts) and *high energy overheads* (sensors such as GPS & gyroscopes have a very high power drain).

To date, transportation analytics has employed either approach in isolation, with little exploration of how these two independent data streams can be intelligently combined to create innovative new insights. Our work in this paper tackles this gap, by demonstrating how these two data streams (infrastructure and mobile) can jointly provide deeper insights into urban transportation and commuting behavior, than currently possible. In particular, we propose a collaborative framework for transportation service analytics, called *TRANSense*, that *(a)* applies spatiotemporal analytics on transportation infrastructural data to detect likely anomalous transportation events (e.g., a high demand for taxis at specific taxi stands), and *(b)* uses such anomalous events to smartly trigger mobile sensing and thereby recognize specific commuting-related

- *Yu Lu is currently with Advanced Innovation Center for Future Education, Beijing Normal University (Email: luyu@bnu.edu.cn). Archan Misra is with the School of Information Systems, Singapore Management University, Singapore. Huayu Wu are with the Institute for Infocomm Research (I2R), A*STAR, Singapore. Wen Sun is with the Xidian University, China, and she is the corresponding author.*

activities of interest (e.g., queuing for taxis) with much higher accuracy and lower energy cost.

To provide practical embodiments of this proposed framework, we shall demonstrate two different analytics applications: (i) a *Taxi Service Analyzer* that detects passenger queuing for taxis and estimates the queuing time at the taxi demand hotspots, and (ii) a *Subway Boarding Analyzer* that identifies events where a commuter fails to board an arriving train (most likely due to overcrowding). Most importantly, we show the **symbiotic benefit** of fusion of these two data sources: *neither of these two services can be readily realized by using a single type of data stream, and the accuracy of recognizing both the underlying infrastructure-based events and individual commuting activities is significantly improved via mutual reinforcement using the other data stream.* Using a combination of small-scale mobile sensing data and currently available "big data" from public transportation information sources in Singapore, we show that these two novel analytics applications can be readily realized with a high fidelity.

Our work in this paper thus makes the following *key contributions*:

- *Unified Framework for Infrastructure+Mobile Sensing:* We propose a novel transportation analytics framework that combines spatiotemporal analytics on large-scale data from different urban transportation information systems with mobile sensing data from a participatory pool of commuters. Using the two services mentioned above as exemplars, we show how such combined analytics can provide hard-to-obtain and *individual-specific* commuting insights with a high accuracy as well as lower energy overheads.
- *Smart Commuter Mobile Sensing:* We adopt a hierarchical decomposition approach to design and implement a sensor-based smartphone application to collect different context information from public transportation commuters. The application supports two triggering mechanisms for energy-efficient sensor data collection: either based on automatic location-driven events (geofences) or remotely by a cloud messaging mechanism.
- *Taxi Service Analyzer:* We describe how this analytics service combines the city-scale taxi trace data with smartphone sensing data to detect passengers queueing for taxis and accordingly estimates the passenger wait time. The key innovations in this system include (i) the use of taxi data to identify taxi demand *hotspots* and further infer the locations where passenger queues *may* be occurring, and (ii) the aggregation of the smartphone-sensing data from multiple passengers to estimate the passenger wait time at such hotspots, and (iii) the use of hotspot-driven mobile sensing triggers to reduce the energy overheads without sacrificing the ability to capture relevant queuing events. Using the month-long trace data of over 20,000 Singapore taxis, together with the participatory sensing data at 6 different taxi stands, we show that this application can (a) detect the queuing activity of commuters with over 90% accuracy, (b) estimate the queuing time at taxi stands within an error of 15%, and (c) impose only minimal overhead (a hypothetical worst-case scenario, where a passenger stays in the vicinity of a taxi hotspot throughout the day, would drain only 4.4% of the smartphone's battery).
- *Subway Boarding Analyzer:* We show how to combine the large-scale entry/exit traces of subway data (captured by tapping of RFID-equipped ticketing card at stations) with the on-platform passenger activity data (captured by the smartphone inertial sensors) to accurately identify the episodes of *failed boarding*: where an individual is unable to get on a train (most likely due to overcrowding). The key innovations include (i) a novel Train Arrival Detection (TAD) algorithm that uses the temporal pattern of passenger exit traces (provided by infrastructure informatics data) to indirectly identify train arrival events, and (b) the fusion of such probabilistic train arrival information with an individual's on-platform activity (captured by mobile inertial sensing) to identify the instances of failed boarding. By combining a subway dataset, which includes 1.7 million commuters and nearly 50 million transactions, with the small-scale mobile sensing studies, we show that this system can identify the failed boarding events with a precision of over 90%, that is more than thrice what is achievable via pure mobile sensing (which often mistakenly translates a passenger's random movement into attempting to board a train).

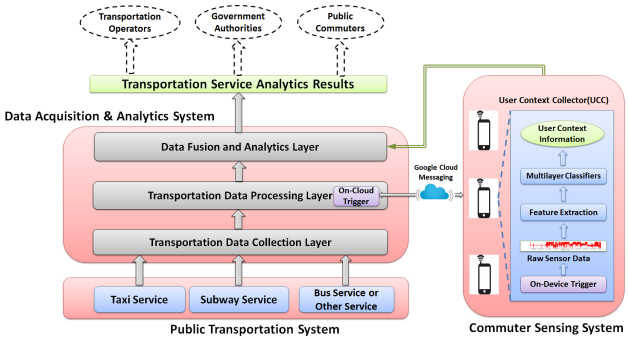


Fig. 1: Block Diagram of the Framework

While our two transport service analyzer applications are both novel, we believe that the main impact of our work is to highlight the broader possibility of creating innovative new personalized transportation services, based on a *combination* of transportation informatics data and commuter participatory sensing.

The rest of the paper is organized as follows: Section 2 depicts the overall system architecture. Section 3 describes the designed commuter sensing tool. In sections 4 and 5, we respectively present the two exemplary applications (*Taxi Service Analyzer* and *Subway Boarding Analyzer*, along with the empirical evaluation results. The discussion and related work are then given in sections 6 and 7. We finally conclude in section 8.

2 System Overview

The block diagram of *TRANSense* is illustrated in Figure 1, which mainly consists of three subsystems, namely *Public Transportation System*, *Commuter Sensing System* and *Data Acquisition & Analytics System*.

2.1 Public Transportation System

This system covers different urban transportation services, such as taxi, subway and bus service. Each service leverages on the corresponding informatics infrastructure to acquire the relevant transportation data. For example, taxi service generates each taxi's real-time GPS location and taxi status, which is collected by the in-vehicle telematics device on taxis. Such individual taxi information can be instantly sent to the backend cloud using

the cellular service. Another example is the subway service that utilizes an RFID-based ticketing infrastructure to record passenger ingress and egress (tap-in and tap-out) information at each subway station. The bus service also collects the passenger boarding and alighting information on each operating bus by leveraging on a similar electronic ticketing system.

2.2 Commuter Sensing System

This system adopts the participatory sensing strategy to capture transportation-relevant context from the smartphones of participating commuters. More specifically, a lightweight application, called *user context collector* (*UCC*), is installed on the smartphones of participating commuters. The application gathers the information from the embedded smartphone sensors (e.g., accelerometer or barometer), and meanwhile performs a high-level context recognition task. The sensor data collection and the context sensing process can be activated and terminated automatically, and thus the entire process does not need any user's manually input or configuration. When installing the UCC application, it is necessary to get the user permission for collecting the sensor data and running the non-intrusive context sensing service.

2.3 Data Acquisition & Analytics System

This system is the central component of the proposed framework to aggregate, fuse and analyze heterogenous data collected from the above described two systems. It adopts a cloud-based three-layer hierarchical architecture, where each layer has its own unique and independent functionality:

- *Transportation Data Collection Layer*: This layer collects and manages the incoming raw and large-scale transportation data from transportation services (e.g., taxi service and subway service). It provides separate interfaces and indexing structures for receiving and managing large volumes of spatiotemporally-indexed data. The transportation data cleaning and preprocessing are also conducted within this layer.
- *Transportation Data Processing Layer*: This layer mainly conducts the analytics on the data from the transportation system, and outputs the intermediate analytics results (e.g., the likely current taxi hotspot locations) to the upper layer. A variety of data mining and analytics techniques can be applied in this layer. Besides, this layer also includes a specifically designed component, called on-cloud trigger, which is used to activate the commuter sensing tasks from the cloud side. The triggering decision is made mainly based on the intermediate analytics results from this layer. Such intermediate results together with the commuter sensing results would be also sent to the upper layer.
- *Data Fusion & Analytics Layer*: This layer firstly aggregates and coordinates the commuter sensing results from smartphones. After that, it applies the appropriate fusion logic to combine the commuter sensing results from smartphones with the intermediate analytics results from the transportation data. Finally, it provides the key analytics and insights for the corresponding transportation service either in an online manner or offline manner.

In short, the above-described three systems in *TRANSense* work cooperatively to acquire, process and analyze the data from both public transportation services and commuters. The final analytics results would possibly benefit different stakeholders, including transportation service providers, relevant government agencies and public commuters.

3 User Context Collector Design

To accurately and effectively collect the commuter context information from a smartphone, we adopt a hierarchical decomposition approach to design and implement a robust and fine-grained application, called the user context collector (UCC).

3.1 UCC Workflow

The UCC mainly consists of an on-smartphone trigger and three classifiers. The on-smartphone trigger is mainly used to automatically trigger the sensor data collection, and then the three classifiers use different feature sets to identify different types of user activity and context. The basic workflow of the UCC is shown in Figure 2, which can be described as below:

1) The installed UCC registers and runs a background service, which periodically fetches the latest location list, such as the current set of taxi demand hotspots or crowded subway stations, from a backend service. If the smartphone's current location is nearby any listed location, the sensor data collection is triggered automatically; otherwise, the UCC sleeps for a specific time period.
2) Once the sensor data collection is triggered, UCC firstly applies a low-pass filter to the raw 3-axis accelerometer measurements and transforms the readings from the phone's coordinates to the world coordinates, i.e., the earth reference frame, by multiplying with the rotation matrix [4], [5].
3) The resulting accelerometer frames are first run through a *kinematic motion classifier*, to detect whether the smartphone is in a "fast movement" state, which is possibly caused by taking a vehicle (subway, bus or taxi) or walking. If the fast movement is not detected, UCC enables the upper two classifiers and meanwhile keeps the kinematic motion classifier running. Whenever a fast movement state is newly detected (indicating that the user has boarded the bus, taxi or train), UCC would notify the cloud side, and then abort the sensor data collection and the classification processes.
4) When no fast movement is detected, the second classifier, called *basic activity classifier*, starts to identify user's basic activity over short-duration time window (e.g., 2 seconds). Using the accelerometer data as the input, the classifier has four output options, namely *stationary*, *stepping*, *walking* and *others*, to label each non-overlapping short-duration time window. The sequence of the classifier outputs serve as the inputs of the upper *advanced activity classifier*.
5) The *advanced activity classifier* performs a more complex and high-level activity recognition task. Our current design is to make the distinction between queuing and non-queuing for each long-term time window (e.g., 120 seconds). If a queuing activity is successfully detected for the current time window, it would be continuously running for the next long-term time window. The classifier currently has two distinct queuing models: the first model is designed for the common "continuous" queuing scenarios, e.g., at taxi stand or supermarket counter, while the second model is for "batched" queuing behavior, e.g., at a subway or bus station. The appropriate model is chosen based on the current location and the corresponding transport service.

Note: While our UCC design is generic, the specific classification models implemented currently conform to the specific applications/use cases that we demonstrate. The development of mobile sensing based commuting activity classifiers is NOT the

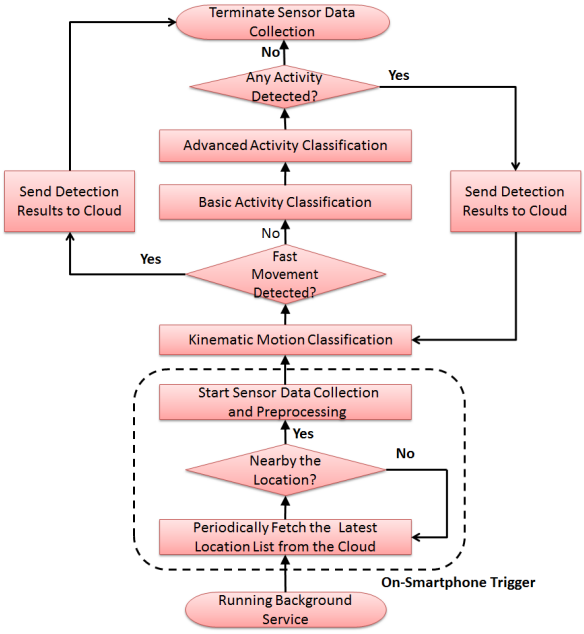


Fig. 2: User Context Collector Workflow

major focus of our work; as needed, the UCC can be extended to capture other activities (e.g., TransitLabel [6] which captures other activities inside a subway station).

As described in the UCC workflow, before building the classification models, the 3-axis acceleration data is first transformed from the phone's coordinates to the earth coordinates using the well-known quaternion-based techniques, and an implementation publicly available on Google developer manual [7]. The transformed acceleration measurements are then independent of phone's orientation, and thus would not be greatly influenced by the phone placement and position.

The three classifiers are not strictly in a hierarchical structure: the *advanced activity classifier* requires the accurate inputs from the *basic action classifier*, and thus its accuracy is indeed influenced by the *basic action classifier*. The *kinematic motion classifier* is mainly used to trigger the system using the detected fast movement, which includes *motorized movement*, *subway movement* and *pedestrian movement*. For example, when the *motorized movement* is wrongly detected as *subway movement*, it will also trigger the system and would not directly influence the accuracy of the upper two classifiers.

3.2 UCC Classifier Design

3.2.1 Basic Feature Extraction

When the UCC application is activated, it firstly transforms 3-axis accelerometer measurements from the phone coordinates's to the world coordinate. After that, the UCC segments the transformed data into non-overlapping fixed-size frames, and computes the following features for each frame:

- *Frequency Domain*: the 5 frequency domain features for the activity recognition, including spectral energy, entropy, peak position, wavelet entropy and wavelet magnitude, and thus a total of 15 features across the 3 axes.
- *Time Domain*: the 10 time domain features including mean, variance, standard deviation, magnitude, correlation, minimum, maximum, range, interquartile range and zero-crossings rate, and thus a total of 30 features across the 3 axes.

3.2.2 Kinematic Motion Classifier

Running at the root of the UCC classifier hierarchy, the kinematic motion classifier is mainly used to detect whether the smartphone user is in a fast movement status or not (i.e., this is, eventually, a binary classifier). The classifier has four intermediate output options: *motorized movement*, *subway movement*, *pedestrian movement* and *others*. *Motorized movement* means user taking motorized transportation (e.g., taxi and bus), *subway movement* means user taking subway train, *pedestrian movement* means user continuously walking. All these three classes are eventually mapped to a "fast movement" label. *Others* means that the user is engaged in some other unknown motion, distinct from "fast movement".

The classifier utilizes all the mentioned frequency domain and time domain features, and thus its feature space consists of 45 features across the 3 axes. The time domain features on the horizontal plane provides the highest discrimination for identifying the accelerations and decelerations that occur when a user is in a vehicle. The time window size for this classifier needs to be relatively large to cover several vehicle acceleration or deceleration periods.

Note that it is also feasible to use alternative methods (e.g., GPS-based methods) for fast movement detection, e.g., periodically estimate the speed using smartphone's GPS data. However, the GPS module consumes much more energy than the accelerometer, and is often unsuitable at several transportation-relatedand locations, such as underground subway stations or taxi stations in a downtown area with with dense buildings and narrow roads (urban canyon effect [8]).

3.2.3 Basic Activity Classifier

The *basic activity classifier* utilizes all the mentioned time domain and frequency domain features as well. The time window size for this classifier needs to be relatively small, as a large time window may cover multiple user basic actions. It would label each time window with one of the four options, i.e., *stationary*, *stepping*, *walking* and *others*. Note that *stepping* means slowly moving forward, which often occurs during queuing. The output labels will be sent to the upper advanced activity classifier as the inputs.

The main difference between the *walking* class in the *basic activity classifier* and the *pedestrian movement* class in the *kinematic motion classifier* is the frame size: *walking* is associated with 2-second frames, whereas the *pedestrian movement* class uses a 20-second frame to capture a relatively long-lived walking activity.

3.2.4 Advanced Activity Classifier

Running at the top level of the UCC classifier hierarchy, the *advanced activity classifier* conducts the high-level user activity distinction. Borrowing from prior work in queuing detection (the QueueVadis system in [9]), the classifier distinguishes between *queuing* and *non-queuing* activity context.

We separately built two models for two different queuing scenarios: (a) model-A is mainly for queuing scenarios where people move forward gradually, such as at a taxi stand or supermarket checkout counter; (b) model-B is for scenarios that people move in a *batched* fashion, such as at subway platform. Both models need to aggregate the consecutive outputs from the lower *basic activity classifier* and then computes the following features over the larger time window: the number of transitions between the four basic actions, the mean and variance of each basic action's time duration. Thus, totally 18 features, which are all derived from the

basic activity classifier outputs, are used in the *advanced activity classifier*. In the evaluation subsection, we shall see that the time window size for this classifier needs to be large enough to capture the characteristics of the queuing activity.

3.2.5 Learning Algorithm

We evaluated several supervised learning models, and finally adopted the decision tree C4.5 for both the kinematic motion classifier and the basic action classifier, and naive Bayes for the user activity classifier. The resulting tree models and the probability models can be interpreted and implemented on smartphones.

3.2.6 Phone Placement and Usage

The phone placement and usage obviously influence the accuracy of the accelerometer-based activity detection. For most queuing/waiting scenarios, we observe that commuters either hold the smartphone on their palm or have it inside a garment pocket or bag; accordingly, we mainly evaluated the classification performance based on such phone placement positions. Similar to the most accelerometer-based activity recognition systems, our classification techniques exhibit poor performance when the phone is subject to usage-induced artefacts, such as the user casually swinging or frequently rotating the smartphone. The most common approach is to simply discard the sensor data when such artefacts are detected.

3.3 Trigger Mechanism from Cloud

Besides actively triggered from the smartphone, the UCC application can also be remotely enabled from the cloud side. In our implementation, such remote triggering is supported via the google cloud messaging (GCM) service (which now supports Android and iOS devices). When a GCM message arrives, it can wake up the device from the sleep state and then activate the specific application using the so-called Intent Broadcast. Figure 3 demonstrates the UCC installation and the cloud-based triggering process.

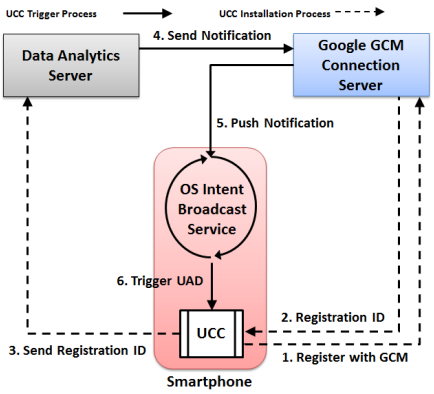


Fig. 3: The GCM Workflow with UCC

Normally, each GCM message sent out by the backend system carries the latest location list, and the enabled UCC would firstly run a background process to retrieve the current device location by using the Google Location APIs [10]. If the device's current location is close enough to any of the location in the list, the UCC would then initiate the sensing and analytics pipeline. Otherwise, the UCC application acknowledges its current location to the server and continues to sleep.

To determine the current location of the device, UCC adopts the Google Location APIs which determines the location by combining different data sources including GPS, cellular signals and WiFi signatures. The GPS module usually imposes a higher energy overheads than others. In practice, we found that most of the located places by UCC are fairly accurate even the GPS module is disabled. It is probably because of the high density of cellular towers and WiFi access points in Singapore.

3.4 UCC Evaluation

We implemented the Android version of the UCC and tested it on Samsung Galaxy S3 and S4. The 3-axis accelerometer sampling frequency is set to 50Hz. The experiment data are collected by multiple participants at 6 different taxi stands and 9 different subway stations under the following natural, but *somewhat-controlled*, scenarios:

- The participant joins a taxi waiting queue, and choose to either walk away or get on a taxi when he reaches the front of the queue. The data are collected by 12 participants during a three-week period.
- The participants join a subway waiting queue on the platform, and either get on the arriving train or continue waiting for the next one due to overcrowding. This data are collected by 12 participants during a three-week period.
- The participants conduct 'random' activities (except queueing) nearby the taxi stands and subway stations, e.g., talking with others, purchasing coffee from Starbucks, or simply walking past the taxi stands or subway stations. The data are collected by 7 participants during a two-week period.
- The participants take taxi, subway and bus as a normal passengers. The data are collected by 6 participants during a two-week period.

For the kinematic motion data (on the train, taxi or bus), each participant collected data daily (mainly during their trips between office and home) over multiple days, for a total duration of 2.1 hours on average. For the basic action data, each participant collected around 1.2 hour data for each of the four actions respectively (i.e., *stationary*, *stepping*, *walking* and *others*). Queuing data was collected for a cumulative duration of approx. 28.6 hours, and involved 39 different queuing episodes at either taxi stands or subway stations. When building the classifiers, we randomly split the training data and testing data into 9:1, and adopt the 10-fold cross-validation approach to evaluate the built models. Moreover, we applied sensitivity analysis to ensure that our models did not suffer from over-fitting.

All the participants are required to record down the ground truth and use any of the three phone positions, i.e., *on palm*, *in trouser pocket* and *in bag*, to ensure the results are not sensitive to the phone placement.

3.4.1 Kinematic Motion Classifier

Table 1 gives the evaluation results of the *kinematic motion classifier*: we see that the F1 scores of *motorized movement* and *pedestrian movement* are both above 0.8, demonstrating that the classifier can well distinguish these two types of fast movement. The F1 score of *subway movement* is sightly lower (around 0.75), which is probably because the subway ride is generally smooth with limited vibrations and jerks than taking car or walking. To further improve the accuracy for the *subway movement* class, the barometer measurement data can be collected and utilized by the UCC [11]. In short, the *kinematic motion classifier* is able to successfully detect the desired fast movement types (*motorized*, *subway* and *pedestrian*), which would be used to obtain the queuing end time and timely terminate the UCC sensor data collection.

TABLE 1: Accuracy of Kinematic Motion Classifier

	Precision (%)	Recall (%)	F1 score
Motorized Movement	86.0	84.1	0.85
Subway Movement	76.2	74.4	0.75
Pedestrian Movement	82.7	87.8	0.86
Others	84.1	82.8	0.83

TABLE 2: Accuracy of Basic Activity Classifier

	Precision (%)	Recall (%)	F1 score
Stationary	87.5	89.4	0.88
Stepping	82.6	80.9	0.82
Walking	87.8	84.3	0.86
Others	80.8	84.0	0.82

By testing different values and evaluating the experiment results, we set the time window size for this classifier to 20 seconds, as such period normally covers several rounds of taxi and subway train accelerations and decelerations.

3.4.2 Basic Action Classifier

Table 2 shows the evaluation results of the *basic action classifier* and we see that the overall classification results achieve a high accuracy (all F1 score above 0.8). The *stepping* class has a relatively low F1 score, but the overall experiment results show that all the desired basic activities can be well identified.

The classifier adopts 2 seconds as its time window size, as a larger time window may include multiple basic actions (e.g., several stepping actions in one time window) and accordingly decreases the classification accuracy.

3.4.3 Advanced Activity Classifier

Table 3 gives the evaluation results of the *advanced activity classifier* in mode-A and mode-B respectively: we see that mode-A achieves a slightly higher accuracy than mode-B, and meanwhile the F1 score of the *queuing* class in mode-B is slightly below 0.8. It is probably because queuing for subway train on platform (mode-B) does not involve many 'typical' stepping/stationary transitions. However, the overall results demonstrate that the classifier can successfully detect the passenger queuing activities in both mode-A and mode-B, i.e., at both taxi stand and subway platform.

The time window for this classifier needs to cover several taxi arrival events or train arrival events. In the practical implementation, the cloud side can periodically compute the average arrival rate of the taxis or subway trains, and notify the UCC the latest value for adjusting the window size.

3.4.4 Energy Consumption

Figure 4 shows the energy consumption of UCC: we see that around 38 milliWatt (mW) is consumed for only sensor data collection, and the value slightly increases to 53 mW after enabling the three classifiers. It indicates that the classifiers impose insignificant energy overhead. The total energy consumption of UCC is around 84 mW, which includes running the GCM service and periodically using the Google Location APIs (GPS disabled). Note that the above values are obtained by using the total energy consumption subtracting the background energy consumption,

TABLE 3: Accuracy of Advanced Activity Classifier

F1 Score	Mode-A	Mode-B
Queuing	0.832	0.794
Non-Queuing	0.890	0.866

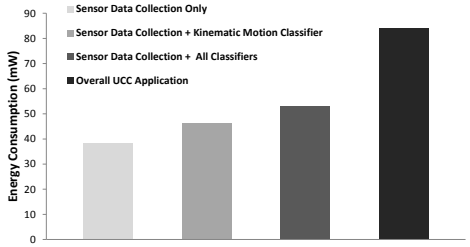


Fig. 4: Energy Consumption of UCC and its Components

where the background energy is measured by keeping the CPU on using the Android WakeLock mechanism[1] and meanwhile minimizing all the default phone services. The Monsoon power monitor[2] is used to conduct all the measurements.

4 Exemplary Application 1: Taxi Service Analyzer

To demonstrate the design, operation and performance of our *TRANSense* framework, we present two exemplary and practical applications: *taxi service analyzer* and *subway boarding analyzer*. In this section, we present the *taxi service analyzer*, with a special focus on how *UCC*-generated individual context information is fused with infrastructural data to detect passenger queues and estimate the wait time.

4.1 Problem Description

Queuing for taxis is an in-escapable fact of life in densely populated Asian cities, such as Singapore and Hongkong. Accurate and real-time estimation of such queuing delays would not only help public commuters make more-informed transportation choices, but also help taxi drivers to find the demand hotspots. Moreover, regulatory authorities (such as the Land Transport Authority (LTA) in Singapore) require such queue information to develop new policies and fare structures. Currently, in Singapore, the LTA conducts daily surveys (published publicly[3]) by manually recording down the taxi passenger waiting time. Our goal is to replace this inefficient manual process with an ubiquitous, smartphone-based approach for both passenger queue detection and queuing delay estimation.

4.2 Transportation Data Collection Layer

Each taxi in Singapore periodically reports their location, status and other critical information to the backend via its in-vehicle device, called mobile device terminal (MDT). By leveraging on the MDT installed on each taxi, more than 20,000 taxis in Singapore keep collecting and updating their real-time states and GPS locations. Such taxi information is then transmitted to a backend system using either general packet radio service (GPRS) or 3G cellular network. The *transportation data collection layer* would collect and buffer such incoming taxi MDT data.

Each MDT message normally contains five important fields: *Taxi State*, *GPS Location*, *Instantaneous Speed*, *Taxi ID* and *Timestamp*. The typical taxi states include *FREE* (available for passenger), *POB* (passenger on board) and *ONCALL* (booked by passenger). Any change of taxi state would trigger a new MDT

1. https://developer.android.com/training/scheduling/wakelock.html
2. https://www.msoon.com/LabEquipment/PowerMonitor/
3. http://www.lta.gov.sg/content/ltaweb/en/public-transport/taxis/taxis-and-the-lta.html

message sent to the backend. Besides, the MDT periodically sends the GPS location update message. We define the important terms and expressions to be used for the taxi data processing.

Definition 1. Individual taxi's trajectory z: A temporally ordered sequence of the MDT messages from one taxi, i.e., $p_1 \rightarrow \cdots \rightarrow p_i \rightarrow \cdots \rightarrow p_n$, where p_i $(1 \leq i \leq n)$ is the MDT records containing the taxi state $p_{i.state}$, instantaneous speed $p_{i.speed}$, latitude coordinate $p_{i.lat}$, longitude coordinate $p_{i.lon}$ and timestamp $p_{i.ts}$.

Definition 2. Multiple taxis' trajectory set Z: A collection of the individual taxi's trajectories, i.e., $\mathrm{Z} = \{z^j | j = 1, 2, \cdots\}$, where z^j is the j^{th} taxi's individual trajectory.

Definition 3. Demand Hotspot spot set Q_{loc}: A collection of locations, i.e., $Q_{loc} = \{q_r | r = 1, 2, \cdots\}$, where q_r is the r^{th} spot witnessing a high demand for taxis.

4.3 Transportation Data Processing Layer

In this layer, the main objective is to firstly determine the taxi demand hotspots, and then infer the hotspots that may have passenger queue (PQ). The taxi demand hotspots are normally the most frequent taxi pickup locations, and PQ can be inferred from the taxi pickup behavior and taxi booking information.

To accurately and timely determine taxi demand hotspots, we propose a practical algorithm, called demand hotspot detection (DHD) algorithm. The basic idea behind the DHD algorithm is that given the taxi trajectory set Z over the buffering duration T_b, we firstly identify the taxi pickup events and their corresponding locations. After that, it uses the density-based clustering method to find out all the frequent taxi pickup locations. The complete algorithm is shown in Algorithm 1.

Algorithm 1 Demand Hotspot Detection Algorithm

Input: All taxi trajectory set Z over buffering duration T_b, speed threshold η_{sp}.
Output: Taxi demand hotspot set Q_{loc} over the buffering duration.

```
k ← 1; R^k ← ∅; G ← ∅;
for each individual taxi's trajectory z in Z do
    for i = 1 → n do
        if p_{i.speed} ≤ η_sp and R^k = ∅ then
            R^k.Add(p_{i−1}); R^k.Add(p_i);
        else if p_{i.speed} ≤ η_sp and R^k ≠ ∅ then
            R^k.Add(p_i);
        else if p_{i.speed} > η_sp and R^k ≠ ∅ then
            R^k.Add(p_i); k ← k + 1; R^k ← ∅;
for each R^k ≠ ∅ do
    if {p_{i.state} in R^k changes from FREE to POB OR changes
    from ONCALL to POB} then
        Add the first POB location to the location set G;
Run DBSCAN clustering algorithm on the set G;
Add the centroid of the found clusters into Q_loc;
```

The DHD algorithm mainly consists of 3 steps: firstly, it extracts multiple sub-trajectories from each individual taxi's trajectory, i.e., $R^1, R^2, \cdots, R^k$, where each sub-trajectory has at least one record with the speed below the given threshold η_{sp}. They are regarded as the pickup event candidates; secondly, it only keeps the sub-trajectory R^k that has the specific pickup patterns, namely the taxi state changes from *FREE* to *POB* or *ONCALL* to *POB*. From each selected R^k, the algorithm takes its first *POB* location after its state transition, and adds it to the location set G. Finally, the algorithm runs the density-based clustering algorithm *DBSCAN* [12] on the location set G, and computes the centroid of each found clusters. These centroids are the identified demand hotspots during buffering duration T_b. Note that we set the buffering duration as a fixed-size moving time window to store and batch process the newly received MDT data.

Based on the identified taxi demand hotspots, the system further infers potential PQ locations using a simple and practical method, called passenger inference (PQI) algorithm. As the inputs of the PQI algorithm, $N_{free}(q_r)$ is the number of arrival taxis at hotspot q_r with *FREE* state, $\bar{t}_{wait}(q_r)$ and $\bar{t}_{dep}(q_r)$ are the average wait time and average departure interval of the arrival taxis at q_r. Besides, $N_{oncall}(q_r)$ is the number of arrival taxis with *ONCALL* state at q_r. The complete PQI algorithm is shown in Algorithm 2.

Algorithm 2 Passenger Queue Inference (PQI) Algorithm

Input: $N_{free}(q_r)$, $\bar{t}_{wait}(q_r)$, $\bar{t}_{dep}(q_r)$, $N_{oncall}(q_r)$, Q_{loc} and thresholds η_{wait},η_{dep},τ_s,τ_l,τ_{ratio}.
Output: Labeled possible PQ with type T_1, T_2 or T_3.

```
for each q_r in Q_loc do
    if N_free(q_r) < τ_s and t̄_wait(q_r) < η_wait then
        Label q_r as a possible PQ location with type T_1;
    else if N_free(q_r) ≥ τ_l and t̄_dep(q_r) < η_dep then
        Label q_r as a possible PQ location with type T_2;
    else if N_oncall(q_r) / N_free(q_r) > τ_ratio then
        Label q_r as a possible PQ location with type T_3;
```

In general, the PQI algorithm uses three separate conditions to infer passengers queuing for taxis:

1) A small number of arrival FREE taxis $N_{free}(q_r)$ and meanwhile a short average taxi wait time $\bar{t}_{wait}(q_r)$ are both observed. This condition indicates that all arrival taxis are quickly taken at hotspot q_r, but the current taxi supply there is relatively small.
2) A large number of arrival FREE taxis $N_{free}(q_r)$ and meanwhile a short average taxi departure interval are both observed. This condition indicates that a taxi queue may currently co-exist with a passenger queue at hotspot q_r.
3) A large ratio of arrival *ONCALL* taxis to *FREE* taxis at hotspot q_r is observed, which means a large fraction of passengers choose to book taxis instead of waiting for FREE ones. This condition is based on the fact that passengers in Singapore usually prefer hailing down a FREE taxi rather than booking one due to significant booking charges.

If any one of the above three conditions is satisfied, the corresponding hotspot would be labeled as possible PQ location, with the type T_1, T_2 or T_3 respectively, during that buffering time period. The DHD and PQI algorithms are designed based on our previous study [13], where we also use taxi status transitions to identify the demand hotspots. A key differentiator in this work is the introduction of a new condition in the PQI algorithm, i.e., the ratio of ONCALL taxis to FREE taxis, to infer PQ.

The latest inferred PQ would be added into the location list on the cloud side, which can be fetched by the running UCC application. Meanwhile, the cloud side actively triggers the UCC application (via GCM messages) on mobile devices that were reported to be near such PQ locations.

4.4 Data Fusion & Analytics Layer

This layer aggregates the queue sensing results from the smartphones and conducts the two analytics tasks: 1) validate the

existence of PQ; 2) estimate the average passenger wait time, denoted as T_{pq}.

When a user 'joining' or 'leaving' a queue event successfully detected, the UCC would send out a message to the cloud side, comprising a 4-tuple payload $\langle id, desc, time, loc\rangle$, where id is the smartphone's GCM ID, $desc$ is either *queuing start* or *queuing end*, $time$ is the current timestamp, and loc is the device location. We design a novel algorithm, called passenger queue validation (PQV) algorithm (see algorithm 3), to process the delivered UCC messages and conduct the two analytics tasks.

Algorithm 3 Passenger Queue Validation (PQV) Algorithm

Input: A new UCC message $\langle id, desc, time, loc\rangle$, hash table W, hotspot set Q_{loc}, threshold η_{dist} and η_q.
Output: Validated passenger queue and avg. wait time T_{pq}.

```
if id is not a key in W then
    for each q_r in Q_loc do
        if distance(loc, q_r) < η_dist AND desc = start then
            Insert a new key-value pair (id, [q_r, time]) into the
            hash table W; N(q_r) ← N(q_r) + 1;
            if N(q_r) ≥ η_q then
                Validate a passenger queue at q_r;
else if id is an existing key in W then
    Get the value [q_r, t_s] from W using id;
    if desc = end AND t_s ≠ null AND time ≠ null then
        Validate a passenger queue at q_r;
        T_pq(q_r) ← (T_pq(q_r) * N(q_r) + time − t_s) / (N(q_r) + 1);
        Remove the key-value pair (id, [q_r, t_s]) from W;
```

Briefly speaking, the algorithm runs in a message driven way, meaning each arrived UCC message would invoke the algorithm once. Moreover, the algorithm constructs a hash table W, whose key is the GCM ID (i.e., *id*) and the value is the user queuing start time t_s together with the corresponding hotspot location q_r. Whenever an UCC message sent from a new smartphone, i.e., id is not an existing key in W, the PQV algorithm would then check whether the location loc is close to any hotspot and whether the UCC message is a "queuing start" message. If both conditions are true, the algorithm would insert a new key-value pair, i.e., $(id,\ [q_r, time])$, into the hash table W, and count one more queuing user at q_r. If the number $N(q_r)$ is more than the threshold η_q, PQV would validate a passenger queue at q_r. The basic design logic is that a passenger queue can be confirmed when several queuing passengers are detected at the same hotspot.

When id is an existing key in W, namely the UCC message sent from a commuter who already starts queuing, the algorithm would directly retrieve the stored values from W, i.e., the queuing start time t_s and the queuing location, say q_r. It then checks whether the delivered message is a "queuing end" message, i.e., $desc = end$. If it is true, PQV would validate the passenger queue at q_r, and meanwhile update the estimated wait time value T_{pq} at q_r. Finally, the existing key-value pair would be removed from the hash table W.

In the practical implementation, when no UCC message arrives from hotspot q_r for more than one buffering duration, all the corresponding variables, including $T_{pq}(q_r)$ and $N(q_r)$, would be initialized. *A key attractive feature of the PQV algorithm is that it does not require a high deployment density–i.e., the the queuing context from a small proportion of waiting passengers can validate passenger queues and estimate average passenger wait time.*

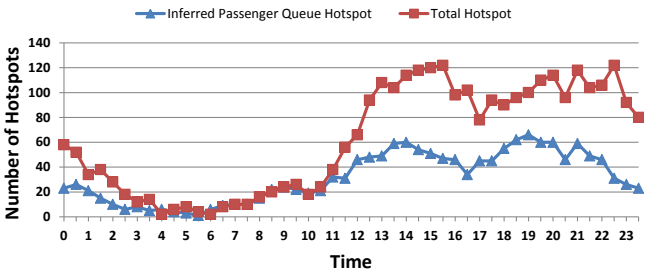


Fig. 5: Variance of Hotspot Number

TABLE 4: PQ Inference Results and Validation

Type Labeled by PQI Algorithm	Percentage	Average Failed Booking Number
T_1	22.9%	4.53
T_2	5.6%	1.45
T_3	10.2%	3.63
Non-Labeled	61.3%	0.66

4.5 Experimental Results

We run the DHD algorithm and the PQI algorithm on the taxi data during one month period: the speed threshold η_{sp} and the buffering duration T_b in DHD algorithm are set to 10 km per hour and 1800 seconds respectively. The thresholds in the PQI algorithm are the localized parameters: for a given hotspot, we compute its 20^{th} percentile of taxi wait time and 20^{th} percentile of taxi departure intervals, and use these two values as the thresholds η_{wait} and η_{dep} respectively; similarly, we compute the 20^{th} percentile and 80^{th} percentile of FREE taxi arrival number during each buffering period, and use the two values as the thresholds τ_s and τ_l respectively; the last threshold τ_{ratio} is set to the average ratio of booking job number to the street job number at the given hotspot.

Figure 5 shows the average number of the identified demand hotspots and the inferred PQ hotspots at different time slots. We see that the total number of the demand hotspots significantly increases during the day time, and less than 50% demand hotspots are inferred as PQ hotspots during the peak hours (11AM to 10PM). Hence, it is not necessary to conduct the smartphone sensing at all the identified hotspots; we can focus on the inferred PQ hotspots. For practical reasons, it is hard for us to concurrently and continually monitor the 120+ possible hotspots in Singapore to collect the corresponding ground truth. Instead, we utilize the manually collected taxi wait times, at 29 designated hotspots, published by Singapore government [14] to perform partial validation of our results. In particular, all the 29 hotspots are correctly detected by the DHD algorithm (thus demonstrating 100% *recall*). and they follow the same time-variant pattern shown in Figure 5.

We select 15 busy demand hotspots, and Table 4 summarizes their PQ inference results: the first column is the labeled type by the PQI algorithm, and the second column is the inferred PQ time in percentage. It shows that averagely 22.9%, 5.6% and 10.2% of the time slots are labeled as T_1, T_2 and T_3 respectively. Meanwhile, 61.3% time slots are not labeled, meaning the nearby participating smartphones would not be triggered then and the corresponding energy cost on smartphones would be saved. Assuming a *worst-case scenario*, where a smartphone is perpetually near a demand hotspot, we see that the UCC on the smartphone would run for approx. 4 hours each day (2 hours in the morning peak and 2 hours in the evening peak). Even in this pessimistic scenario, the UCC's daily energy consumption is approx. 350 milliWatt-hours, i.e., around 4.4% of the nominal battery capacity of a Samsung

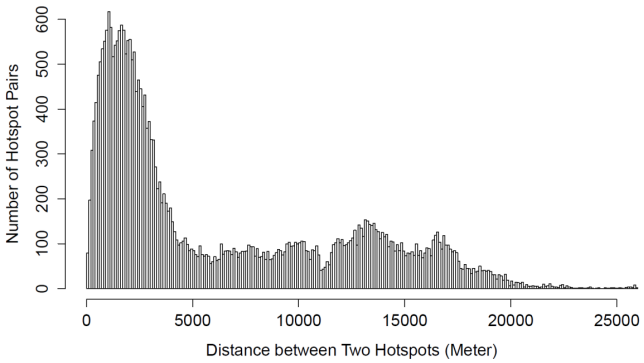
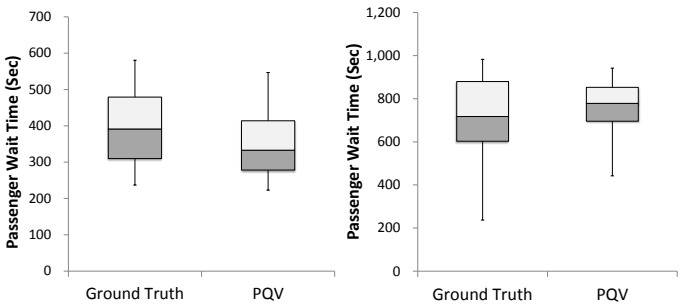


Fig. 6: Histogram of Distance between Two Hotspots



(a) Morning Sessions (b) Evening Sessions

Fig. 7: Estimation of Passenger Wait Time

smartphone (7.98Wh).

The third column in Table 4 is the corresponding failed booking number: a failed booking means a taxi booking request dispatched to the taxis nearby but no one available, and thus a large failed booking number at a hotspot indicates the current taxi supply less than its demand here. We see that the non-labeled type has a much smaller failed booking number, i.e., 0.66, than the other three types, which at least to some extent validates the PQ inference results. The type T_2 has a relatively lower number (i.e.,1.45) than type T_1 and T_3. It is possibly due to T_2 is derived from the condition that a large number of FREE arrival taxis with a fast departure rate: where passengers see more FREE taxis and thus are more willing to wait in the queue rather than booking one.

Figure 6 shows the histogram of the distance between any two identified demand hotspots: only 4 pairs have the distance smaller than 50 meters, and the majority falls in the range of 1000 to 5000 meters. Given the GPS localization error can be several meters [15], it shows that most identified demand hotspots are relatively far from each other, and thus the commodity smartphones with GPS module are able to well localize and distinguish these demand hotspots. The identified demand hotspots include not only the taxi stands, but also other popular pickup locations, such as hospitals and schools.

We conduct the passenger queue validation experiments during both morning and evening peak hours at 6 different busy hotspots, which consists of totally 31 sessions and each session operates at least 30 minutes. During the session, 3 to 5 participants randomly join the PQ with their own smartphones running the UCC application. The smartphones are put into the participant's trouser pocket or holding on the palm. An observer takes down the ground truth during each session: the start and the end time of each queuing people (including both the participants and other queuing passengers). The commercial 3G cellular networks are used to communicate with the backend cloud. On the cloud side, the distance threshold η_{dist} and the parameter η_q used in the PQV algorithm are set to 100 meters and 3 respectively. Meanwhile, all the UCC applications periodically fetch the location list from the cloud every 3 minutes.

Among all the sessions, 90.3% sessions successfully validate the PQ at the given hotspot. The unsuccessful cases are mainly due to two reasons: 1) the queuing time is too short to be captured by the UCC application (typically smaller than 90 seconds); 2) the UCC application misclassifies the queuing activity into the *non-queuing* activity. Figure 7 shows the box plots of the passenger wait time from the PQV algorithm and the ground truth collected by the observer during both the morning sessions and evening sessions. We see that the system can fairly estimate the passenger wait time with the average mean error less than 15%. Note that the box plot results are based on the fact that at least 3 participants with UCC in the same passenger queue, while the estimation error may increase if fewer participants conduct sensing there. In short, the experiment results show that the system can accurately validate PQ and meanwhile fairly estimate the passenger wait time at the hotspots.

Fig. 8: Queuing Passengers inside a Subway Station

5 Exemplary Application 2: Subway Boarding Analyzer

In this section, we present the second application, which collaboratively uses personal smartphone sensing data in tandem with records of passenger ingress/exit activity at subway stations.

5.1 Problem Description

Due to the popularity of the subway system (which provides a reliable and extensive transportation network that is immune to traffic jams or bad weather), there is an extremely high demand, especially during peak hours. (Fig. 8 shows a snapshot of queuing passengers inside a Singapore subway station.) As a result, trains often get filled beyond their maximum loading capacity, and queuing commuters at subsequent stops may not be able to board an arriving train, but instead have to wait for the subsequent train on the platform. Such "failed boarding" (FB) events are not uncommon at the busy stations in Singapore, and contribute significantly to a lowered perception of the overall commuting experience.

To capture such individual-specific FB events, we require information from both an individual commuter and the subway operations infrastructure. Currently, the subway stations use automatic ticketing gantries, which require commuters to tap in and tap out their RFID-equipped tickets, thereby providing a repository of timestamped entry/exit traces of passenger data. By leveraging on such large-scale subway informatics records and individual-level smartphone sensing, our *TRANSense* framework can enumerate the severity of FB events at different stations/platforms.

TABLE 5: Field and Description of Ticketing Card Record

Field	Description
Card_Number	Encrypted Card ID
Entry_DT	Date and Time when Passenger Entering Station
Exit_DT	Date and Time when Passenger Exiting Station
Origin	Location ID of Entry Station
Destination	Location ID of Exit Station

5.2 Transportation Data Collection Layer

In Singapore, the subway system, called massive rapid transit (MRT) system, serves more than one million commuters daily [16]. The deployed automatic ticketing system uses a contactless ticketing card to charge the trip fares at all subway stations. As ticketing card users get a fare discount, nearly all Singaporean residents (and the vast majority of visitors) use the card to utilize MRT and bus services. For the MRT service, the ticketing card is required to tap in and tap out at the gantry to calculate the current trip fare. Each card transaction record in the system contains multiple fields; for our FB-analytics work, we use only the fields (which are completely anonymous) summarized in Table 5.

5.3 Transportation Data Processing Layer

This layer has two objectives (both based on the ticketing card data): (a) the main objective is to detect the subway train arrival events on the platform, while (b) a secondary objective is to identify the likely "crowded" subway stations (this identification being used to trigger the sensing/analytics by the UCC client on nearby mobile devices).

Train Arrival Detection: While the subway trains nominally have a fixed arrival time schedule, in practice, they are easily delayed due to longer boarding time or other unexpected situations, especially during the peak hours. Even a single train's short delay at a station results in cascaded impacts on multiple subsequent operating trains and their arrival time. We thus design a practical algorithm, called *train arrival detection* (**TAD**), to detect the actual train arrival time using the ticketing card data. The complete algorithm is shown in Algorithm 4 and 5.

The basic idea is as follows: shortly after the arrival of a train at a busy station, alighting passengers would lead to a *burst* of tap-out transactions at the station. Therefore, such train arrival events can be detected by discovering such short-lived deviations in the overall tap-out rate in the ticketing card transactional data. In our current implementation, the large volume of ticketing card data is maintained in a Hadoop system, and analyzed using the MapReduce framework. The Mapper essentially groups all the ticketing card records of an individual destination station. A Reducer will firstly uses the given MRT station ID to find all the exiting passengers at the station during the given time window W. It then classifies the exiting passengers into different traveling directions by utilizing their origin station ID. (Note: the need to obtain the traveling direction arises because we would like to distinguish between the arrival of trains on the two different directions of travel associated with each train line.) After that, the algorithm counts the number of departing passengers separately (for each direction) during the current time window, e.g., N_A , and the next time window, e.g., N'_A. Finally, the algorithm identifies the train "arrival events" using the following two conditions: 1) a significant increase from N_A to N'_A; 2) no train arrival event is detected during the last time window at that direction. In addition, Δt is deducted from the marked time window W' at the final step, to account for the time that a passenger takes to walk from the platform to the station exit gantry. Δt is clearly station dependent (it mainly depends on the internal layout of the station) and typically ranges between 60-180 seconds in our studies.

Algorithm 4 Map Function of the TAD algorithm

Input: An empty *key*, and a card transaction record *value*
1: **if** *value* is valid **then**
2: Set *key* = *value*.Destination;
3: Emit(*key*, *value*);

Algorithm 5 Reduce Function of the TAD algorithm

Input: A *key* of a station ID, and an iterator *values*, threshold θ.
1: Initiate empty lists Q;
2: **while** *values.hasNext()* **do**
3: *Q.add(values.next())*;
4: Sort Q based on *Exit_DT*, denoted as $\{ez(i)|i = 1, 2, \cdots\}$;
5: Set the sliding time window W, and $N_A \leftarrow 0$, $N_B \leftarrow 0$;
6: **for** all $ez(i).Exit\_DT$ in W **do**
7: **if** $ez(i).Origin \in Direction\ A$ **then**
8: $N_A \leftarrow N_A + 1$;
9: **else if** $ez(i).Origin \in Direction\ B$ **then**
10: $N_B \leftarrow N_B + 1$;
11: Slide to next time window W';
12: Repeat the above steps to obtain N'_A and N'_B during W';
13: **if** $\frac{N_A{}' - N_A}{N_A} > \theta$ AND W is not marked for Direction A **then**
14: Mark W' for Direction A as a train arrival time window;
15: **if** $\frac{N_B{}' - N_B}{N_B} > \theta$ AND W is not marked for Direction B **then**
16: Mark W' for Direction B as a train arrival time window;
17: emit(*key*, marked time window $W' - \Delta t$);

In the TAD algorithm, we use the "origin station ID" to help determine the train's direction. In reality, some origin stations cannot clearly indicate the passenger's travelling direction, as multiple feasible (and likely) routes (on different lines) may exist between the observed (origin, destination) pair. However, for each direction at a given destination station, it is possible to find a smaller set of "unambiguous" origin stations, which provide a clear indication of the passenger travel direction. Such an "unambiguous" origin station has two properties: 1) either there is only one feasible route from it to the destination; 2) or there may be multiple routes, but one route has a significantly shorter traveling time than other alternatives. In the practical implementation of the TAD algorithm, for any given destination station, the system only choose the ticketing card records generated by such "unambiguous" origin stations. We do observe that the number of such "unambiguous" stations is much lower at key interchange stations (where multiple lines intersect). As a consequence, the overall volume of ticketing card records processed by the TAD algorithm may be much smaller than the total volume of exiting passengers, thereby potentially impacting the accuracy of our analysis. In such cases, the system may need to utilize additional information from the ticketing system: for example, the entry/exit gantry ID would be helpful, as passengers with different subway lines/directions usually use different gantries. For our current paper, we perform the analysis with TAD only on those stations that have a sufficient number of "unambiguous" origin stations (and thus do not require any additional gantry-level information).

The threshold θ affects the accuracy of train arrival detection. An overly large value fails to detect the actual train arrival events, while an unduly small value leads to false alarm (too many train arrival events). Moreover, an appropriate value for θ also depends on each station's popularity (the intensity of departing or arriving

passengers at different times). Hence, our approach models θ as a station-specific parameter; we shall explain the empirical derivation of θ later, when we describe the experimental results. Note that, in the future, θ may also be continually adjusted by automated machine learning techniques that effectively attempt to match the inter-arrival distribution and total number of train arrivals to known values obtained from external observations (e.g., known values of train frequencies).

Crowded Station Identification (CSI): A separate CSI algorithm is used to process the underlying ticketing card transactional records to identify the set of crowded stations–i.e., the ones where there is likely to be a higher incidence of FB events. The basic idea is as follows: a crowded station usually has a large mismatch between the number of entry passengers and exit passengers, with the number of entry passengers far exceeding the exiting ones. The core logic of the CSI algorithm (represented as the Reduce operation of a separate Map-Reduce implementation) is shown in Algorithm 6.

Algorithm 6 Crowded Station Identification (CSI) Algorithm

Input: Grouped card records $\{ez(i)|i = 1, 2, \cdots\}$, station ID s_j and thresholds η.
Output: Marked crowded subway station.

```
Set sliding time window W, and N_entry ← 0, N_exit ← 0;
for each ez(i) do
    if ez(i).Origin = s_j AND ez(i).Entry_DT ∈ W then
        N_entry ← N_entry + 1;
    if ez(i).Destination = s_j AND ez(i).Exit_DT ∈ W then
        N_exit ← N_exit + 1;
if (N_entry − N_exit) / (length of W) > η then
    Mark s_j as a crowded station during time window W;
```

Similar to the TAD algorithm, the mapper function groups all the ticketing card records by origin station and destination station, and sends them to a reducer to process. The CSI algorithm on the reducer side operates on a per-station basis, first calculating the difference between the corresponding entry rate and departure rate. If the mismatch is larger than the threshold value, the algorithm will mark the station *crowded*. As a result, the *TRANSense* framework would then trigger the UCC applications (in a manner similar to the Taxi Analyzer application) on nearby smartphones to activate the sensing needed to detect potential FB events. The threshold η in the CSI algorithm is critical: An overly large value would result the system failing to detect the crowded stations, while an unduly small value would lead to a number of false alarms. Hence, it is carefully set as a station-specific parameter as well, which will be elaborated in the evaluation part.

5.4 Data Fusion & Analytics Layer

In this layer, the main objective is to detect FB events by combining the detected train arrival events and the smartphone sensing results from the UCC applications. The designed UCC application can detect the passenger queuing activity on the platform (by mode-B of the *advanced activity classifier*) and the *subway movement* (by its *kinematic motion classifier*)–i.e., it can detect both queuing-related behavior by a passenger on the platform as well as the subsequent *subway movement* state of a passenger who has successfully boarded the train. More specifically, the UCC classifier would notify the server of a 4-tuple payload $\langle id, desc, time, loc\rangle$, where id is the smartphone's GCM ID, $desc$ is either *queuing* or *subway movement*, $time$ is the current timestamp, and loc is the device's location.

We now present the failed boarding detection (FBD) algorithm that identifies such FB events by combining such mobile device-generated alerts with the train arrival events detected from the ticketing card data. The complete one is shown in algorithm 7.

Algorithm 7 Failed Boarding Detection (FBD) Algorithm

Input: A new UCC message $\langle id, desc, time, loc\rangle$, hash table H_T, station set S and the threshold η_{dist}.
Output: Detected FB events.

```
if id is not a key in H_T then
    for each station s_i in S do
        if distance(loc, s_i) < η_dist AND desc = queuing then
            Insert a new key-value pair (id, [s_i, time]) into H_T;
else if id is an existing key in H_T then
    Get the value [s_i, t_s] from H_T using the id;
    if desc = subway movement then
        Get the moving direction, say A, using loc and s_i;
        if any train arrival detected during [t_s, time] in A then
            A new FB event detected at station s_i in direction A;
            Remove the key-value pair (id, [s_i, t_s]) from H_T;
```

Similar to the PQV algorithm used in the taxi service, the FBD algorithm also runs in a message driven way–i,.e., each arriving UCC message would invoke the algorithm once. It constructs a hash table H_T, whose key is the GCM ID of smartphones and value is the queuing time t_s together with the subway station s_i. Whenever an UCC message sent from a new smartphone, i.e., id is not a key in the hash table H_T, FBD would then check whether the smartphone location loc is close enough to a subway station and whether the UCC message is a "queuing" message. If both conditions are true, the algorithm would insert a new key-value pair, i.e., $(id,\ [s_i, time])$, into the hash table H_T.

Whenever the arrived id is an existing key in H_T, meaning the UCC message is sent from a smartphone whose user is already in the "queuing" state, FBD would directly retrieve the stored value from H_T, i.e., subway station s_i and queuing time t_s. If the UCC message is a "*subway movement*" message, the algorithm will get the smartphone's moving direction based on its latest location and its queuing station. However, if any train arrival in the same direction is detected during the time period for which the user was in the 'queuing' state, the algorithm would confirm a new FB event detected (for that specific user) at that station. The existing key-value pair would be removed from the hash table H_T.

Note that the proposed algorithm identifies the FB events of only the participating smartphone users. It may thus appear that this system is capable only of providing individual-specific insight about the commuting experience. However, in reality, multiple commuters typically the same "skip the current train" behavior when faced with an overcrowded train. Accordingly, it is not necessary to capture FB events from all commuters on a platform. The detection of multiple FB events from even a smaller set of participatory users can alert the train operator that the train has become overcrowded. Accordingly, for this particular application, the *TRANSense* framework is capable of providing useful aggregate-level insights as well.

5.5 Experimental Results

We ran the TAD and CSI algorithms on our one-month long subway dataset (which included nearly 50 million transaction

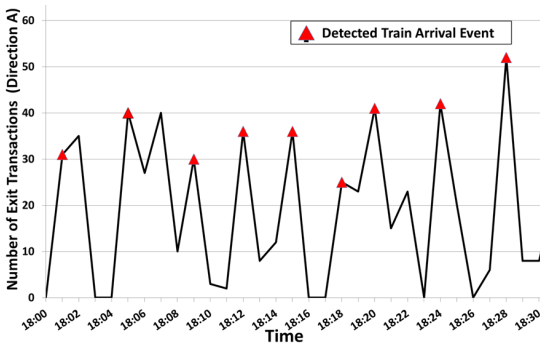


Fig. 9: Detected Train Arrival Events using Ticketing Card Data

records from 1.7 million subway commuters). The sliding time window size W is set to 60 seconds and 900 seconds for the TAD algorithm and CSI algorithm respectively. The parameters θ used in the TAD algorithm and η used in the CSI algorithm are computed empirically for each individual MRT station as follows. For a given station, we collect its exit passenger numbers at each time window and then compute the 60^{th} percentile over all the positive increasing rates (i.e., windows where the passenger exit exceeds that of the prior window). The computed value is used as the threshold θ for that station. Separately, we calculate the difference of entry rate and exit rate during each of "net inflow" time windows (i.e., time windows where the entry passenger count exceeds the exit count), and then use the 80^{th} percentile over all such differences as the threshold η for that station.

TAD & CSI Performance:

We run the TAD algorithm at the busy stations during morning peak (7:30AM–9:30AM) and evening peak (17:30PM–19:30PM) periods. Our results show that 86.7% train arrival events are successfully detected. TAD fails in two distinct scenarios: 1) when two consecutive trains arrive within a short time interval; or 2) only a few passengers alight from the arrival train (and thus do not cause a blip in the MRT exit records). Fig. 9 shows an example of applying TAD at a busy station during the evening peak hour: we see that during the 30-minute time period, 9 train arrival events are successfully detected, with the average inter-train arrival gap of approx. 200 seconds. TAD failed to detect only one train arrival (the event at 18:07PM), as its arrival time was quite close to the previous one (only around 100 seconds time interval).

We run the CSI algorithm on the same period of the ticketing card data: of the total set of subway stations, we find that 11 and 13 subway stations, respectively, are frequently (more than 75% of the time) marked as "crowded' stations during the morning peak and the evening peak hours. During the morning peak, the total hourly entry rate is around 65000 and the exit rate is around 33000–i,e., on average, around 1084 passengers enter and 617 passengers leave these 11 stations every minute.

FB Detection Performance: To study the ability to accurately detect FB events, we conducted multiple experiments, during both morning peak and evening peak hours, at 9 different subway stations (consists of of a total of 22 distinct sessions). During each session, 3 to 4 participants joined the waiting queue on the platform with the smartphone running the UCC application. The smartphones were either in the participant's trouser pocket or held in their hand. A separate observer noted the ground truth of each participant (his or her 'queuing start' and 'train boarding' times) during the session, as well as the arrival and departure time of each train on the platform. The commercial 3G cellular networks are used to communicate with the backend cloud. On the cloud side, the distance threshold η_{dist} used in the FBD algorithm is set to 100 meters. All the participating UCC applications periodically fetch the crowded station list from the cloud every 5 minutes. Note that one FB event means one participant failed to board the latest arrival train; for some sessions, we may have no FB event (as all participants successfully boarded the first arriving train).

TABLE 6: Failed Boarding Detection Results

	Detected Number	Ground Truth	Percentage
Morning Peak Hours	27	31	87.1%
Evening Peak Hours	26	29	89.7%
Total	53	60	88.3%

TABLE 7: Accuracy of Two Methods for FB Detection

FB Detection Method	Precision (%)	Recall (%)	F1 score
Subway Data + Smartphone Sensing	94.6	88.3	0.91
Smartphone Sensing Only	32.5	91.6	0.48

Table 6 summarizes the FB event detection results: we see that 87.1% and 89.7% FB events are successfully captured during the morning session and evening sessions respectively. We also found that nearly 30% of the detected FB events occurred *consecutively*–implying that the participants failed to board multiple consecutively-arriving trains due to overcrowding. On the other hand, around 11.7% FB events are not detected–such false negatives were mainly caused by the UCC classification errors in the kinematic motion classifier and advanced activity classifier.

To clearly understand the need for combining such MRT and mobile sensing data, we compared our approach with an alternative *smartphone-only* method, where only smartphone sensing was used to detect the FB events. In this approach, we assumed that an FB event occurred whenever a queuing commuter's activity (on the platform), as computed by *basic activity classifier* in UCC, was seen to transition either (i) from *stationary* to *stepping* or (ii) from emphstationary. Table 7 gives the FB detection results for both the *TRANSense* and *smartphone-only* approaches: we see that both methods achieve a good recall rate, implying that most of actual FB events can be successfully detected by both methods. However, the precision of the second method (smartphone sensing only) is significantly lower, i.e., only around 32.5%, due to a large number of false positive cases. In particular, in the absence of corroborating train arrival data, random movements of a commuter are interpreted incorrectly as FB events, leading to a high false-positive rate.

In summary, the experimental results show that the *TRANSense* framework can accurately capture the FB events for the subway service, and that the collaborative fusion of the subway data with the smartphone sensing data significantly increases the FB detection accuracy.

6 Discussion

6.1 Possible Limitations

In general, our algorithms are based on the expected behavior of commuters at *typical* high-demand hotspots (for both taxis and train stations), and may need to be extended to handle less-common scenarios. For example, the *Taxi Service Analyzer* assumes that a large queuing time occurs primarily at hotspots (those locations with a significant number of taxi pickup events). However, there may be certain remote passenger queuing locations, where the overall taxi demand is low but taxi arrivals are even more intermittent. While it is likely that passengers may simply book taxis at such locations, the long wait time is still unavoidable, and thus new algorithms and design may be needed for such cases. Similarly, the *Subway Boarding Analyzer* assumes that the

arrival of a crowded train at a platform will usually result in a noticeable number of disembarkations and consequent passenger 'tap-out' transactions. While it is a reasonable assumption, this approach of detecting train arrival events will fail for some special stations (through which most passengers simply transit without leaving the station). A more careful analysis of the passenger entry/exit patterns at upstream and downstream stations (that needs to effectively take into account the connectivity graph of the train network) might be able to reveal such 'hidden crowding' cases.

Some time-dependent and location-dependent parameters used in the current algorithms, such as the thresholds used in Algorithm 5 and 6, are crucial to the system performance. Currently, these parameters are derived from empirical observations using a primarily *heuristic approach*. To provide greater robustness, we are now exploring the use of machine learning techniques to automatically derive these values based on historical measurements. For example, the thresholds can be learnt via simple adaptation techniques that aim to maximize the correlation or agreement between sampled ground-truth values and inferences obtained via our algorithms. Moreover, cost-sensitive classification techniques can be used to appropriately reflect the application-specific desired tradeoff between false-positive and false-negative errors.

6.2 Offline vs Real-time Implementation

The empirical results presented in this paper are all based on offline analytics–using traces of historical subway and taxi data, and replaying them through the developed *TRANSense* modules. While such offline analytics is adequate for our goals (of demonstrating the unique ways in which transactional transport data can be combined with participatory mobile sensing), additional modest system enhancements may be needed to support a more real-time implementation. For example, the detection algorithms for taxi hotspot locations or crowded stations may need to modified to support more aggressive and early detection; similarly, the Map-Reduce based implementation of our analytics may need to be enhanced to support asynchronous (potentially delayed) arrival of transactional data. Such "systems" issues provide interesting directions for future work.

6.3 Smartphone Sensing Enhancement

In the current implementation of the UCC application, we do not make a distinction between car movement and bus movement in the kinematic motion classifier and simply regard both as the *motorized movement* class. It is feasible to further classify these two transport modes in the next version of UCC: such design would help the *Taxi Service Analyzer* to easily exclude the bus passengers at the locations where taxi demand hotspots are nearby the bus stops. More features and sensor data may need to be added, such as using the audio sensor [17] to detect the beep sound of the ticketing card reader on the bus. Besides, the UCC application may need to keep running for additional several minutes, especially after commuters getting on the vehicle, to accurately identify the transportation mode. Besides, the bus-related commuter sensing information, combing with the public bus data, would enable building a new analyzer specifically for the bus service.

6.4 Transportation Data Enrichment

In the given two exemplary applications, we only use the *basic* information of the transportation data from the taxi system and subway system. The richer and better quality of transportation data would help to significantly improve the system performance. Take the current *Subway Boarding Analyzer* as an example: it is relatively hard to accurately detect the train arrivals at some interchange stations by only using the origin and destination station ID, as such origin-destination pairs cannot determine a unique train route and direction. However, it would be much easier to tackle this issue, when the system further utilizes the exit gantry ID, as subway commuters taking different subway lines normally pass through different gantries at the interchange stations.

Furthermore, the transportation data used in the current two applications are mainly collected from the public transport operators. The latest technologies, e.g., near field communication (NFC), would enable the smartphone-based payment for the transportation service, and thus the smartphone may have both transportation service information and user activity information. It would enable a new way to aggregate the information from both sides, and accordingly facilitate building new analytics applications for the public transportation services.

6.5 Potential Applications

The core of the proposed *TRANSense* framework is the belief that combining data from transportation-related infrastructural information sources with personal mobile sensing can provide richer *person-centric insights* that go beyond simple operational statistics. Such insights can benefit both the commuting public and transportation operators. For example, accurate and real-time estimation of queuing delays at bus stations and taxi stands can be integrated into a travel-planning mobile application, which a commuter can use to make better real-time decisions (e.g., deciding whether to wait for a taxi or take a bus from a nearby location). Accurate and fine-grained understanding of a commuter's transportation-related experiences can also help operators engage better with consumers. For example, if the application detects multiple failed-boarding attempts, it can provide dynamic rewards to specific commuters, thus helping assuage some of their justified resentment. In a similar vein, if an application can accurately detect how long an individual elderly commuter had to stand in a bus before getting a seat (as opposed to just computing overall crowdedness levels), operators could preferentially reward such commuters with incentives (e.g., higher chances of winning lucky draws) or more intelligently deploy small autonomous bus fleets on targeted short-distance routes (thereby tackling localized aggravations that are hard to infer purely from infrastructural data).

Note also that, while the specific two applications described here were driven by the characteristics of the Singaporean transportation network, the generic *TRANSense* concepts can apply to other cities, which may have their own distinct characteristics. For example, cities such as London have a flat fare system for buses, and hence cannot track passenger disembarkations–in such scenarios, *TRANSense* can provide useful fine-grained observations on a commuter's behavior.

7 Related Work

7.1 Urban Computing and Transport Data

Taking advantage of the vast amounts of mobile data generated from heterogeneous sources in urban space, urban computing [18] principally focuses on quantifying key aggregate-level metrics for improved city operations. As part of these efforts, public transportation data, including taxi data [19] and smart card data [2], have been widely used to tackle a variety of urban problems. For example, the taxi data are used to study urban planning [20], traffic mobility [21] and the driver recommender systems [22].

The smart card data are used for travel behavior analysis [23], passenger segmentation [24] and station density study [25]. We utilize such city-scale datasets in tandem with modest amounts of participatory mobile sensing to uncover other a commuter's *latent* and *personalized* experiences with transportation services.

A number of cities are now making transportation related datasets more publicly available. For example, Copenhagen provides a platform called *city data exchange* [26] to publish its bicycling and other transportation data. London [27] and Los Angels [28] also provide similar platforms and transportation datasets to support urban computing and data analytics.

7.2 Mobile Phone Sensing

The camera-based solutions [29] for human activity recognition usually suffer the complexity and the privacy issues [30], and the solutions using wearable sensors can well address such issues, especially for the classification of everyday locomotive activities (such as sitting, standing and walking) [31]. On the other hand, the sensor-based activity classification techniques exhibit poor performance when the sensor is subject to usage-induced artefacts, such as the user casually swinging or frequently rotating. The most common approach is to simply discard the sensor data when such artefacts are detected.

Smartphone-based sensing and activity recognition [32], including analysis of user queuing behavior [9], have made rapid progress recently. Smartphone sensor data has also been used to classify different modes of vehicular transport, including bus, car and trains in [33]. These studies utilize different features extracted from a variety of smartphone sensors, especially from the 3-axis accelerometers. Alternative sensing modalities have also been used to infer urban commuting states, e.g., Anderson and Muller [34] explored the use of cellular signal variation to capture different types of user movement. Our UCC application utilizes the same inertial-sensing based estimation strategy; however, we innovate in developing an energy-efficient triggering strategy and a 3-tier classification architecture that is specially tuned to our commuting activities of interest.

7.3 Participatory Sensing

Participatory sensing [3] approaches rely on the voluntary contribution of mobile sensing data from multiple participants to infer various urban environmental states. Zhou *et al*. [17] utilize mobile phone's cellular signal and audio information from bus passengers to predict bus arrival time. Ganti *et al*. [35] gather vehicles' on-board diagnostics and location information to study the fuel efficiency for different routes. Consolvo *et al*. [36] collect individual's nutrition and exercise information to track their personal fitness. More recently, Elhamshary *et al*. [6] have developed the TransitLabel system to annotate various semantic locations at train stations, based on large-scale participatory sensor data.

8 Conclusion

We have introduced the *TRANSense* framework, which aims to derive insights by fusing aggregate insights from city-scale transportation informatics data sources with carefully-activated participatory mobile sensing data. We demonstrated the promise of this approach via two applications. First, the Taxi Service Analyzer is able to identify taxi demand hotspots and then employ mobile sensing to estimate the wait time of individual commuters (within an error bound of 15%, and with practically negligible energy overhead). Second, the Subway Boarding Analyzer is able to reliably (with over 90% precision) identify the failed boarding attempts that commuters make at crowded stations–this accuracy is achieved by ingeniously detecting train arrival events from smartcard transactional records, and imposes low energy overheads by triggering mobile sensing only during instants when such boarding activity is plausible.

On a broader canvas, the *TRANSense* framework demonstrates the importance of combining mobile sensing and infrastructural transaction data: the insights obtained reach a *level of accuracy* that purely participatory mobile sensing has struggled to provide. We believe that many other insights of practical interest (e.g., quantifying how crammed a commuter feels in a bus, capturing how many passengers share a taxi ride) can be estimated using this framework. Moreover, our results show that certain aggregate insights (e.g., the average waiting time at a taxi queue) can be obtained even with very low levels of participatory sensing by the commuters. In ongoing work, we are working to deploy real-time versions of our analytics platform, for adoption by a large-scale commuter population.

References

[1] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. Urban computing with taxicabs. In *Proc. ACM Conference on Ubiquitous Computing (UbiComp)*, New York, NY, USA, 2011.

[2] Marie-Pier Pelletier, Martin Trpanier, and Catherine Morency. Smart card data use in public transit: A literature review. *Transportation Research Part C: Emerging Technologies*, 19(4):557 – 568, 2011.

[3] D. Estrin, K. M. Chandy, R. M. Young, et al. Participatory sensing: applications and architecture [internet predictions]. *IEEE Internet Computing*, 14(1):12–42, 2010.

[4] W. Premerlani and P. Bizard. Direction Cosine Matrix IMU: Theory. Techical Report, 2009.

[5] Yunus Emre Ustev, Ozlem Durmaz Incel, and Cem Ersoy. User, device and orientation independent human activity recognition on mobile phones: challenges and a proposal. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (UbiComp)*, pages 1427–1436. ACM, 2013.

[6] Moustafa Elhamshary, Moustafa Youssef, Akira Uchiyama, Hirozumi Yamaguchi, and Teruo Higashino. Transitlabel: A crowdsensing system for automatic labeling of transit stations semantics. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, 2016.

[7] Android Developer Manual. [online] Available: https://developer.android.com/reference/android/.../SensorManager.html.

[8] M. Dottling, F. Kuchen, and W. Wiesbeck. Deterministic modeling of the street canyon effect in urban micro and pico cells. In *Proc. IEEE International Conference on Communications (ICC)*, pages 36–40, June 1997.

[9] Tadashi Okoshi, Yu Lu, Youngki Lee, Rajesh Krishna Balan, and Archan Misra. Queuevadis: Queuing analytics using smartphones. In *Proc. ACM Conference on Information Processing in Sensor Networks (IPSN)*, 2015.

[10] Google Location APIs. [online] Available: https://developer.android.com/google/play-services/location.html.

[11] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L. Ananda, Mun Choon Chan, and Li-Shiuan Peh. Using mobile phone barometer for low-power transportation context detection. In *Proc. ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2014.

[12] Martin Ester, Hans peter Kriegel, Jrg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[13] Yu Lu, Shili Xiang, and Wei Wu. Taxi queue, passenger queue or no queue? - a queue detection and analysis system using taxi state transition. In *Proc. International Conference on Extending Database Technology (EDBT)*, Belgium, 2015.

[14] LTA Annual Report 2014. [online] Available: https://www.lta.gov.sg/content/ltaweb/en.html.

[15] N. M. Drawil, H. M. Amar, and O. A. Basir. Gps localization accuracy classification: A context-based approach. *IEEE Transactions on Intelligent Transportation Systems*, 14(1):262–273, March 2013.

[16] Erika Fille Legara, Christopher Monterola, Kee Khoon Lee, and Gih Guang Hung. Critical capacity, travel time delays and travel time distribution of rapid mass transit systems. *Physica A*, 406:100–106, 2014.

[17] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait? predicting bus arrival time with mobile phone based participatory sensing. *IEEE Transactions on Mobile Computing*, 2013.

[18] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: Concepts, methodologies, and applications. *ACM Transaction on Intelligent Sys. and Tech.*, 2014.

[19] Pablo Samuel Castro, Daqing Zhang, Chao Chen, Shijian Li, and Gang Pan. From taxi gps traces to social and community dynamics: A survey. *ACM Comput. Surv.*, 46(2):17:1–17:34, December 2013.

[20] Nicholas Jing Yuan, Yu Zheng, Xing Xie, Yingzi Wang, Kai Zheng, and Hui Xiong. Discovering urban functional zones using latent activity trajectories. *IEEE Transactions on Knowledge and Data Engineering*, March 2015.

[21] Javed Aslam, Sejoon Lim, Xinghao Pan, and Daniela Rus. City-scale traffic estimation from a roving sensor network. In *Proc. ACM Conference on Embedded Netw. Sensor Sys. (SenSys)*, 2012.

[22] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. A cost-effective recommender system for taxi drivers. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2014.

[23] F. Zhang, J. Zhao, C. Tian, C. Xu, X. Liu, and L. Rao. Spatiotemporal segmentation of metro trips using smart card data. *IEEE Transactions on Vehicular Technology*, 65(3):1137–1149, 2016.

[24] L. M. Kieu, A. Bhaskar, and E. Chung. Passenger segmentation using smart card data. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1537–1548, 2015.

[25] J. Zhang, X. Yu, C. Tian, F. Zhang, L. Tu, and C. Xu. Analyzing passenger density for public bus: Inference of crowdedness and evaluation of scheduling choices. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2015–2022, 2014.

[26] City Data Exchange. [online] Available: https://www.citydataexchange.com.

[27] London TFL Data. [online] Available: https://tfl.gov.uk/info-for/open-data-users/.

[28] LADOT Traffic Counts. [online] Available: https://data.lacity.org/A-Livable-and-Sustainable-City/LADOT-Traffic-Counts-Summary/94wu-3ps3.

[29] Joshua Candamo, Matthew Shreve, Dmitry B Goldgof, Deborah B Sapper, and Rangachar Kasturi. Understanding transit scenes: A survey on human behavior-recognition algorithms. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):206–224, 2010.

[30] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials*, 15(3):1192–1209, 2013.

[31] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D Lane, Tanzeem Choudhury, and Andrew T Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM conference on embedded networked sensor systems*, pages 71–84. ACM, 2010.

[32] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, 2011.

[33] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proc. ACM Conference on Embedded Network Sensor Systems(SenSys)*. ACM Press, 2013.

[34] Ian Anderson and Henk Muller. Practical context awareness for gsm cell phones. In *Proceedings of the 10th International Symposium on Wearable Computers*, ISWC '06, 2006.

[35] Raghu K. Ganti, Nam Pham, Hossein Ahmadi, Saurabh Nangia, and Tarek F. Abdelzaher. Greengps: A participatory sensing fuel-efficient maps application. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[36] Sunny Consolvo et al. Designing for healthy lifestyles: Design considerations for mobile technologies to encourage consumer health and wellness. *Found. Trends Hum.-Comput. Interact.*, 6, 2014.

Yu Lu received the Ph.D. degree in computer engineering from National University of Singapore (NUS) in 2012. He currently serves as the chief scientist at Advanced Innovation Center for Future Education, Beijing Normal University (BNU). Before joining BNU, he was a research scientist at A*STAR, Singapore. His research interests include data analytics, ubiquitous computing and learning technology.

Archan Misra is a Professor and the Associate Dean of Research in the School of Information Systems (SIS) at Singapore Management University. He also directs the LiveLabs and CASA research centers at SMU, which focus on innovations in mobile systems and smart city technologies respectively. His current research interests are in the areas of wearable & IoT sensing, real-time socio-physical urban analytics and mobile crowdsourcing. Archan received his Ph.D. from University of Maryland at College Park in 2000.

Wen Sun is currently an associate professor with the School of Cyber Engineering, Xidian University. She received her Ph.D. degree in Electrical and Computer Engineering from National University of Singapore in 2014. Her research interests cover a wide range of areas including body sensor network, IoT, participatory sensing, and 5G.

Huayu Wu received his Ph.D. degree in computer science from the School of Computing, National University of Singapore in 2011. He is currently the head of the Data Management and Optimization Lab, Institute of Infocomm Research, A*STAR, Singapore. He has research interests in general database and data analytics topics. He is also actively engaged in applied research.