Collaborative Recognition of Queuing Behavior on Mobile Phones

Qiang Li, *Student Member, IEEE*, Qi Han, *Member, IEEE*, and Limin Sun, *Member, IEEE*

Abstract—Nowadays people spend a substantial amount of time waiting in different places such as supermarkets and amusement parks. Detecting the status of queuing may benefit both users and business. In this paper, we present QueueSense, a queuing recognition system to assist in a queue management system. QueueSense consists of clients on smartphones that provide automatic, energy-efficient, and accurate queuing recognition, and a server in the cloud that collects data, identifies multi-queue lines, and provides waiting time estimation. In order to be useful, QueueSense should be able to recognize queuing behavior in various queuing scenarios without greatly decreasing the battery life of mobile phones. We present features of queuing and build the classifier on smartphones to automatically recognize queue classifier without human input. We investigate the complicated nature of energy consumption for queue recognition on phones and design an effective algorithm to maximize energy savings while guaranteeing accuracy of queue recognition. We evaluate QueueSense performance using the data set from real world queuing scenarios collected over a three-month period. Empirical results show that QueueSense is adaptive to various queuing scenarios with both high recognition accuracy and energy efficiency. We further implemented a prototype of QueueSense, the first queue detection system using smartphones. We conducted real-world experiments in a dining hall and a supermarket near a university campus. Through implementation and evaluation, we demonstrate that QueueSense is capable of detecting waiting lines that occur in our daily lives.

Index Terms—Smartphone sensing, queuing group behavior, queue waiting time prediction

1 Introduction

Queuing is a pervasive phenomenon occurring in public areas such as supermarkets, restaurants, banks, transportation terminals, and amusement parks. People spend a substantial amount of time waiting in lines, and long waiting time brings about awful user experience. Before determining whether to take a line or not, people usually want to know more about the waiting lines, e.g., the number of people ahead or waiting time. Providing such queuing information helps customers better spend time doing something alternative rather than blindly waiting in line, thereby mitigating their anxiety and improving user experience. In addition, managing queues is important for business since it can help reduce inefficient resource allocation and revenue loss. Therefore, there is a need for a better understanding of emerging trends of the queues in order to not only improve user experience but also benefit business.

Nowadays, mobile phones present an attractive platform for people-centric applications [1] as they integrate a variety of sensors that may make similar observations as human. Most smartphones [2], [3], [4] have built-in sensors that measure motion, orientation, location and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy for sensing and inferring queuing behavior recognition. Besides, we observe that people often carry their phones when they are not home and people are willing to obtain queue information if it does not affect the typical usage of their smartphones. This motivates us to use commodity mobile phones to detect queuing and obtain queuing information as a people-centric sensing application.

While the idea of using smartphones for queuing detection seems simple, many challenges arise in practice. First, queuing is a temporal group activity that is different from individual human activity recognition (HAR) [5]. A group behavior cannot be fully understood by piecemeal observations [6]. Second, queues are sensitive to and affected by environments. Queue recognition should be robust to handle various queue scenarios. Last, continually monitoring queuing context incurs significant overhead in terms of energy consumption on mobile phones, which has impacted daily usage of mobile phones.

In this paper, we propose a collaborative approach for queuing recognition based on mobile phones by following queuing rules First Come First Serve (FCFS) that users obey in the line. The key insight is that queuers have similar and relatively stable motions and directions than non-queuers in terms of their individual temporal activities. This queuing behavior characteristic can operate in conjunction with machine learning tools [7] for determining whether the user is queuing or not. We also refer to this approach as low-infrastructure since it only requires the collaboration from users' phones and can be more easily retrofitted in existing queuing application scenarios without any manual effort or the infrastructure support. We also study energy consumption for queue recognition on smartphones at different levels—individual activity sensing, and collaborative detection of queues. An effective algorithmic solution is

- *Q. Li and L. Sun are with the Beijing Key Laboratory of IOT Information Security Technology, and the Institute of Information Engineering, Chinese Academy of Sciences. E-mail: {liqiang43, sunlimin}@iie.ac.cn.*
- *Q. Han is with the Department of EECS, Colorado School of Mines, Golden, CO 80401. E-mail: qhan@mines.edu.*

Manuscript received 1 June 2014; revised 13 Jan. 2015; accepted 6 Feb. 2015. Date of publication 26 Feb. 2015; date of current version 1 Dec. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2015.2407400

developed to maximize energy savings while guaranteeing accuracy of queue recognition.

We have implemented a prototype of QueueSense with clients on smartphones based on Android platforms and a server in the cloud. Smartphones use widely available sensors such as accelerometer, compass and bluetooth to sense individual activities. Queuing features are computed based on queuing characteristics in term of individual activities and support vector machine (SVM) is used to automatically detect whether people are queuing or not on mobile phones. The cloud backend processes multi-lines scenarios and provides estimation of queue length and waiting time. Agglomerative hierarchical clustering is implemented on the server to divide queuers into different lines based on changing rate of relative positions of queuers.

We validate the effectiveness of our approach using the data set collected from real world settings over a three month period, which consists of 40 queuing experiments in three scenarios: dining hall, supermarket, and amusement park. Experimental results show that queuing behavior recognition has achieved high precision and recall. We evaluate the energy consumption on queuing behavior recognition of our adaptive algorithmic solution. We further deploy the prototype of QueueSense on the mobile phones of 10 people and these people participated in waiting in lines in a dining hall and a supermarket near a university campus. Experimental results show that its performance is acceptable for a people-centric sensing application on mobile phones.

Briefly speaking, we make the following contributions in this work.

1) We develop practical solutions to automatically recognize queuing behaviors without manual input and identify multiple concurrent lines.
2) We address energy consumption in queuing detection on mobile phones in three stages: sensing, communicating and computing. An adaptive algorithm is developed to adjustment collaborative sensing and inferring for queuing detection.
3) We implement a prototype application based on QueueSense on commodity phones.
4) We conduct empirical studies of QueueSense in real world scenarios. Results show that QueueSense has achieved high accuracy.

2 System Design Considerations

In this section, we discuss the technical considerations that underpin the design of QueueSense in two aspects: queuing recognition in various scenarios and energy limitations on mobile phones.

Directly using smartphones sensors to sample raw data to recognize queuing behavior faces two challenges. First, queuing is a group activity different from individual activities. When people are waiting in line, they often sequentially perform several simple activities such as "standing"/"sitting" and "walking". Even a person is not waiting in line, these activities may also occur. Therefore, it is hard to conceptually distinguish queuing from non-queuing purely based on individual activities. Second, queuing behavior of users are largely influenced by queuing style and environment. In different environments, queuing behavior is

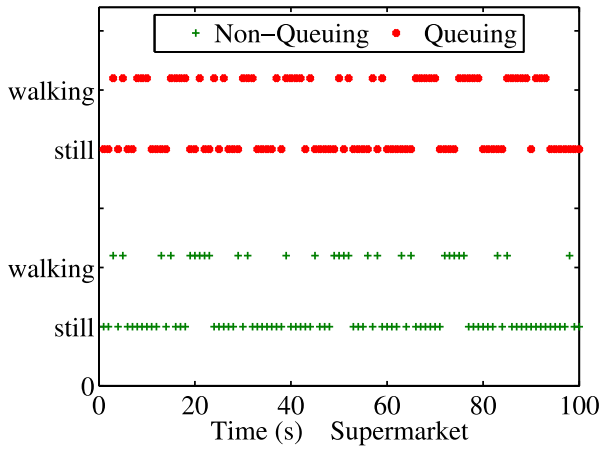


Fig. 1. In a supermarket, people's individual activity changes over time.

depicted by different characteristic of the activities. It leads to different waiting time and movement distance for queuers at every service interval. For instance, queuers waiting at an amusement park move forward tens of meters each time, and queuers at the checkout of a supermarket move forward about half of meter each time. The method of collecting queuing behavior in large annotated environments and then training generic classifiers for all the possible models is simply not feasible nor, we argue, necessary.

To get a sense of the practical challenges that queuing recognition presents to the design of QueueSense on a mobile phone, we conducted a controlled experiment using smartphones. Fig. 1 shows the changes of individual activities over time for one queuing user and one non-queuing user in a supermarket close to the Information Engineer Institute of Chinese Academy of Sciences (IIE-CAS). For simplicity, we use "walking/still" to represent the changes of individual activities in queuing and non-queuing (such as eating at a table, shopping in a store, or walking at a amusement park). We observe that even in the same queuing scenario, there is no difference in the snapshot of individual activities between queuers and non-queuers. Further, since service time may greatly impact queuing behavior of users, we investigated the service time between different queuing scenarios. Fig. 2 shows that queuing service time in the supermarket is only one third of that in the amusement park. We conclude that for different queueing circumstances, changes in individual

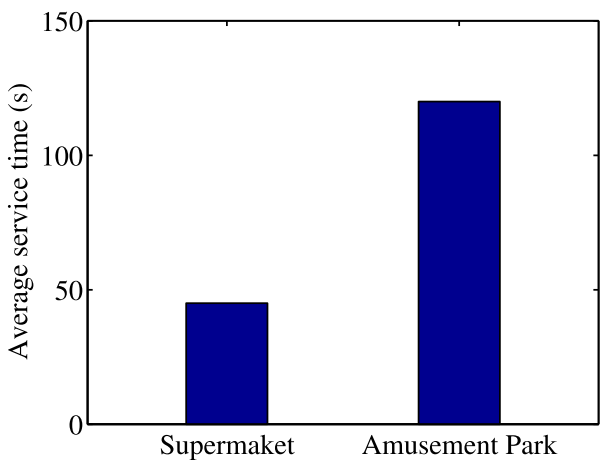


Fig. 2. Average service time in two different queuing scenarios.

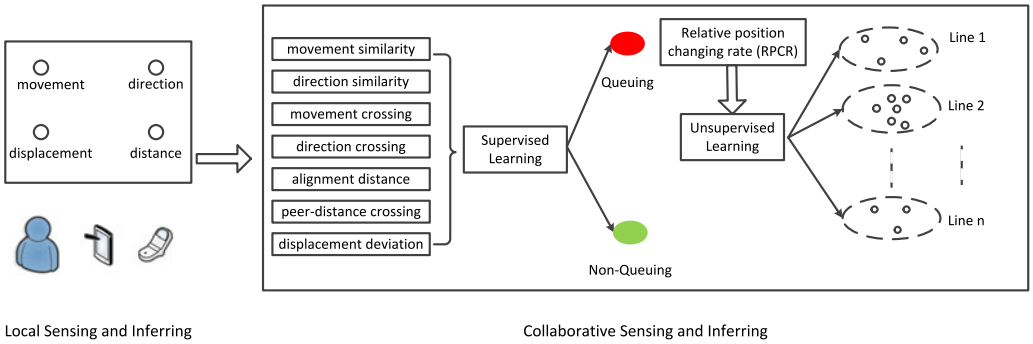


Fig. 3. Software architecture of queue recognition.

activities is inconsistent since different scenarios have different service rate. In brief, the results of individual activities changes during queuing clearly suggest that individual activities is not an viable feature for recognizing queuing behavior.

In this paper, we present a practical solution for recognizing queuing. Our solution is based on Kurt Lewin's "*Field Theory*" [6] which states that group behavior is a function of the attributes, characteristics, and social relationships of all involved parties. We next present the general notion of a queue as the function of multi-queuers according to the overall characteristics [8]: if the front of a line has come to a stop, the back has to stop too; if a person in a queue begins to move forward, then people behind him will do the same. This notion provides the insights necessary for distinguishing people who are queueing and who are not. Our formal definition of queueing is as follows.

Definition 1 (Queue Notion). *Queuing is the behavior that people comply with the rule of first come first serve (FCFS), where people have similar mobility and the sequential relationship of the mobility is stable.*

Another pressing challenge when designing a continuous sensing application such as QueueSense is that mobile phones are resource limited. In particular, a big hurdle for context sensing is the limited battery capacity of mobile devices. In this paper, we explore energy consumed in queuing recognition and then design an effective algorithm to maximize energy savings while guaranteeing accuracy of queue recognition.

3 Queue Recognition

In this section, we first present an overview of QueueSense and then discuss the detailed techniques to tackle the challenges laid out previously, queuing recognition on mobile phones.

3.1 Overview

QueueSense (Fig. 3) consists of two stages: local sensing and inference, and collaborative sensing and inference. In the following, we describe the details of each stage and the interactions between them.

Local sensing and inferring. We mainly compute four features of individual activities: movement, direction, displacement, and distance. These features are used later to extract queuing pattern features. The advantage of using these results instead of raw sensor data is the significant reduction of data that need to be communicated and thus energy saved. The four individual features including movement, direction, displacement, and distance are computed using multi-modality sensors already built in commodity mobile phones: accelerometer, compass, and bluetooth. As individual activity recognition is not the focus of this work, we defer the details of computing these four features to Section 5.

Collaborative sensing and inferring. We divide this stage into two steps: queuing behavior recognition and queue partition. i) The selection of queuing features is critical in building a classifier. To capture the characteristic of queuing behavior, we use seven features derived from individual activities. These features are chosen because they are robust to various scenarios and they can be used to compute similarity between individuals. First, according to the *queuing notion*, queuers have similar sequential mobility. This is captured by four features including movement similarity, direction similarity, movement crossing rate (MCR), and direction crossing rate (DCR). Second, queuers follow the rule: FCFS. Hence, we adopt peer-distance crossing rate (PDCR) to capture whether they are following the rule. Third, people in the front of a line move forward first; and people in the back move later. To capture this, we use alignment distance (AD) to reflect temporal relationship between queuers. QueueSense adopts a supervised learning algorithm for queuing activity recognition. ii) After queuing behavior is recognized on mobile phones, QueueSense adopts an unsupervised learning algorithm to partition queues. Specifically, we use changing rate of relative positions as the distance metric for hierarchical clustering.

In the rest of this section, we present the details of queuing activity recognition and queue partition, the core of QueueSense.

3.2 Queuing Behavior Recognition

As shown in Fig. 3, queuing behavior recognition consists of two steps: extraction of features that can distinguish queuing from non-queuing, and running a classification model on the extracted features.

3.2.1 Queuing Feature Extraction

Queuing features are extracted from four features of individual activities including movement, direction, displacement, and distance computed by mobile phones in advance.

a. Movement similarity (MS). We use the activity overlapping degree to measure movement similarity among people. It is a time-domain feature that reflects two persons' movement similarity in a fixed time interval. Typically, queuers wait when the line stops and queuers sequentially move forwards when the line starts. Hence, queuers have similar movement. MS between person a and person b is computed as follows, assuming each person has a list of n movement activities s_i in the time interval,

$$MS(a,b) = \frac{1}{n}\sum_{i=1}^{n} f(a,b)_i, \tag{1}$$

where

$$f(a,b)_i = 1 \quad if \quad s_i^b = s_i^a. \tag{2}$$

$MS(a,b) = 1$ if and only if the movements of a and b are exactly the same in the given time interval.

b. Direction similarity (DS). We also use activity overlapping degree to measure direction similarity among people, so DS is computed as follows in a way similar to how MS is computed, except that individual direction is typically continuous but movement is discrete,

$$DS(a,b) = \frac{1}{n}\sum_{i=1}^{n} g(a,b)_i, \tag{3}$$

where

$$g(a,b)_i = 1 \quad if \quad avg(h_i^a) - avg(h_i^b) \le \frac{std(h_i^a) + std(h_i^b)}{2} + G, \tag{4}$$

where h_i is a personal direction reading, avg is the average and std is the standard deviation of individual direction readings in the time interval, and G is a guard factor. Basically, two persons are considered to have similar directions if the difference in their average compass readings is no higher than the average standard deviations. When two persons have similar directions, they are more likely to be in the same line. This heuristic method is based on [9].

c. Movement crossing rate (MCR). Crossing rate is usually defined as the number of time-domain crossings within a time duration. We use it to represent the changing rate of movement between people. When people are waiting in line, they "move" forward sequentially as the line moves forward and they "stop" when the line stops. This implies that during the same time interval, the queuers have the same movement. MCR is computed as follows:

$$MCR(a,b) = \frac{\sum_{i=1}^{n-1} |f(a,b)_i - f(a,b)_{i+1}|}{n-1}, \tag{5}$$

where $f(a,b)_i$ is calculated using Equation (2). If two persons a and b have the same movements in two successive time steps, 0 is added to MCR; otherwise 1 is added. Queues typically have regular intermittent movement, namely stop to wait and then start to move forward. If the time interval is long enough to cover the time of the changing movement, MCR reflects at what similarity degree the movements remain stable in the given time interval.

d. Direction crossing rate (DCR). Direction crossing rate reflects the stability of queuers' directions during queuing. It is calculated similarly to Equation (5), except that we replace $MCR(a,b)$ with $DCR(a,b)$, and replace the function g using Equation (4) to determine whether two persons have similar directions.

e. Displacement deviation (DD). It is observed that people who are queuing have similar displacement. We use feature displacement deviation to capture displacement similarity among people. DD is computed as follows:

$$DD(a,b) = \frac{\sum_{i=1}^{n} f'(a,b)_i}{n}, \tag{6}$$

where $f'(a,b)_i$ is the difference between displacement of persons a and b at time step i. It is a time-domain feature that reflects two persons' average displacement difference in a fixed time interval.

f. Peer-distance crossing rate (PDCR). It is observed that when people are queuing, their peer distances remain stable. We use peer distance changing rate of people to distinguish between queuing and non-queuing behavior. PDCR between persons a and b is computed as follows, assuming each person has a list of n peer-distances $d(a,b)_i$:

$$PDCR(a,b) = \frac{\sum_{i=1}^{n-1} (d(a,b)_i - d(a,b)_{i+1})^2}{n-1}. \tag{7}$$

When people are queuing, their peer distances remain stable; thus PDCR is low.

g. Alignment distance (AD). In time series analysis, alignment distance measures similarity between two temporal sequences. It basically calculates the distance between two time-domain sequences in the fixed time interval by counting the minimum number of operations needed to transform one list into the other, where an operation is defined as an insertion, a deletion, a substitution, or a transposition. Similarity in two movement sequences can be represented by AD, even if one queuer in the front of the line moves forwards first and the queuer in the back of the line moves later. For queuing behavior recognition, we compute AD for the three types of temporal sequences: direction, movement and peer-distance. Specifically, we apply dynamic time warping (DTW) to measure AD.

3.2.2 Queuing Classification

We build a classification model for determining whether people are queuing or not, using the seven queuing features previously computed. As shown in Fig. 3, the model is a binary classifier that its output takes only two values, queuing or non-queuing. Given a data point, the queuing classifier is to decide which class the point should be in, queuing or non-queuing. The seven queuing features are viewed as a nine-dimensional vector, including MS, DS, MCR, DCR, DD, PDCR, AD×3. Considering the limitations of mobile phones, a classifier should only require lightweight processing and meanwhile can work in various real world scenarios. Various machine learning algorithms [7] can be used, including logistic regression (LA), linear discriminant analysis (LDA), naive Bayes classifier (NB), support vector machine, and decision trees (DT). Our experiments show that these five classification models do not have substantial

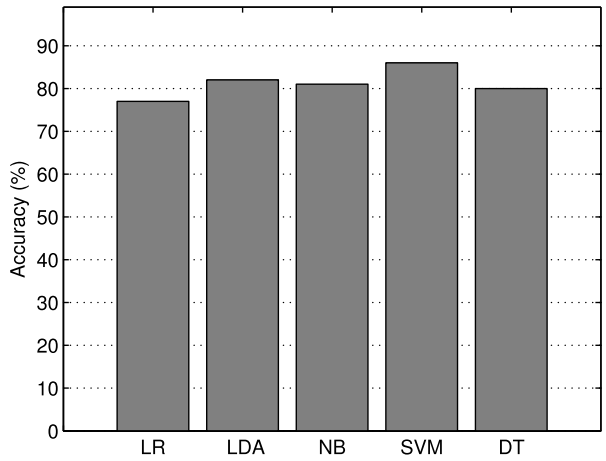


Fig. 4. Performance of various classification models for queuing behavior recognition.

performance differences (Fig. 4). We decide to use SVM since it can generate a maximum margin classifier.

3.3 Queue Partition

Multiple lines usually appear in public service areas. Even queuing behavior recognition achieves high accuracy, inaccurate classification of queuers into different lines will inevitably lead to inaccuracy in estimation of queue length and waiting time. Therefore, a fine-grained partitioning is required to divide queuers into the right lines.

We apply agglomerative hierarchical clustering [7]. It works iteratively as follows. Two queuers with the smallest distance are joined by a branch satisfying a given distance metric. These two queuers are then removed from the set of queuers, and the new branch is added into the set of queuers as a new queuer. We use the average linkage to compute the distance between a new queuer and other remaining queuers. This process is repeated until no queuers satisfy the distance criterion. Each cluster then represents one queue.

A key issue in using this hierarchical clustering technique is the choice of an appropriate distance metric. Apparently Euclidean distance is inappropriate as it does not capture the disparity of queuers in different lines. We address this issue based on such an intuition. When queuers are in the same line, their relative positions would remain stable unless one of them leaves this line. This is because typically people are complying with the rules of waiting in lines. When queuers are in different lines, there is no guarantee that their relative positions remain stable. People who are in different lines have a higher value of relative position changing rate (RPCR) than people who are in the same line.

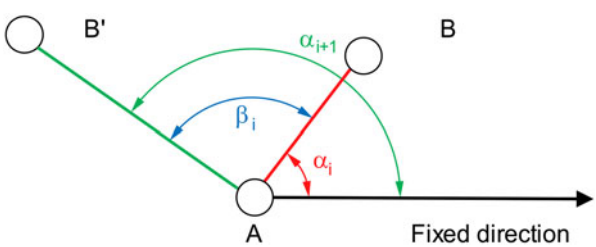


Fig. 5. The angle β_i indicates whether two persons have changed their relative position or not in two successive time slots.

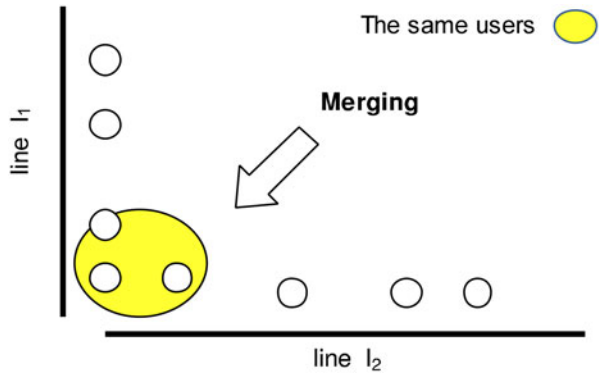


Fig. 6. Two straight lines ($l1$ and $l2$) have the same users at the turn, so they are merged into one line.

Therefore, We use queuers' relative position changing rate as the distance metric for clustering.

We calculate the RPCR feature as follows. Let us consider the example shown in Fig. 5. Persons A and B are at the locations marked as labels at time i, so their peer distance is $|AB|$. α_i is the angle between the vector $\overline{AB}$ and a fixed direction. At time $i+1$, B moves to location B', the angle becomes α_{i+1}, the peer distance becomes $|AB'|$, and $|BB'|$ is the displacement. The difference angle β_i (i.e., $\beta_i = |\alpha_{i+1} - \alpha_i|$) can be calculated using the Cosine theorem of triangle:

$$\beta_i = \arccos\left(\frac{|AB|^2 + |AB'|^2 - |BB'|^2}{2 \cdot |AB| \cdot |AB'|}\right). \tag{8}$$

The change in the β can be used to determine changes in the relative position between A and B. We calculate $RPCR$ between two queuers A and B using β in a fixed time interval as follows:

$$RPCR(a,b) = \frac{\sum_{i=1}^{n-1} |\beta_i - \beta_{i+1}|}{n-1}. \tag{9}$$

If two queuers are in the same line, their relative angle β would remain stable and $RPCR$ is small.

The feature $RPCR$ used in hierarchical clustering can identify straight lines. In practice, lines may not always be straight. They can be zigzagging or having a turn. To handle these cases, we use a merging technique. More specifically, if two straight lines have the same users at one end of the line and the two lines have different directions, we merge them into one line (i.e., one cluster). Fig. 6 shows an example of this case where L1 and L2 are merged into one line. Further, when the coverage range of a queue is larger than the one-hop direct communication range of phones, merging lines can increase coverage range.

4 Energy Efficiency Mechanism

In this section, we first analyze energy consumption of QueueSense on smartphones, then present an adaptive scheme to reduce energy consumption. The energy efficiency scheme can be extended to general group behaviors recognition.

4.1 Energy Consumption Analysis of QueueSense

In QueueSense, energy is consumed by two stages: local and collaborative recognition. Each stage has three tasks: raw data sampling, features extracting, and classification model

TABLE 1
Energy Consumption of Different Sensors on Smartphones

Sensor Type	*Energy*
Accelerometer	21 mW
Magnetometer (Compass)	31 mW
Gyroscope	70 mW
GPS sampling	210 mW
Microphone	180 mW

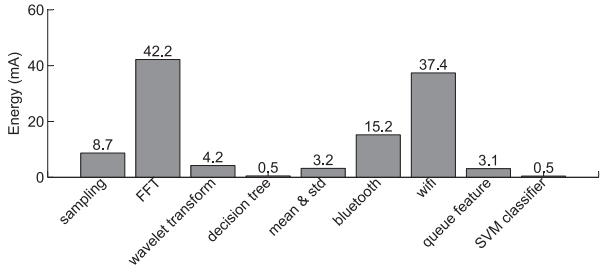


Fig. 7. The current drawn by each component in QueueSense.

computing. Raw data captured is transformed into features, allowing learning algorithms to recognize either individual activity or queue group activity. We next analyze energy consumption of these three tasks.

Raw data collection. In local sensing, energy consumed by raw data collection is impacted by two factors: sensing and sampling duration. The first factor varies by specific sensor type. Different sensors consume different amount of energy in sensing. We have evaluated power consumption using empirical power models constructed from measurements in Power Tutor [10]. Table 1 shows energy consumption of different sensors on smartphones. For instance, GPS costs more than accelerometer at an order of a magnitude. The second factor is sampling *duty cycles*, i.e., whether periodic sampling and sleeping, or continuous sampling. Energy savings on raw data sampling can be achieved by carefully selecting less energy consuming sensor type and their *duty cycles*. Previous work such as [11] has investigated how to achieve energy efficiency on raw data sampling, so we can use their approach for QueueSense.

In collaborative sensing, energy consumed by raw data collection is essentially communication cost for receiving the individual activity list from neighbors. Similarly, there are also two factors impacting energy consumption on raw data sampling: networking cost and the time interval between two successive communications. Detailed analysis of energy cost of different communication methods may be found in [12]. The frequency of communication is an important part for energy consumption of QueueSense. An adaptive scheme for adjusting intervals between communications should not be limited to queuing recognition; instead, it should be easily extended to other group behavior recognition.

Feature extraction. In QueueSense, the same procedure of feature extraction is used for local recognition and collaborative recognition. They both adopt signal processing techniques to extract features for the classification model in the next stage. The energy cost depends the computing overhead of signal processing in terms of smartphone CPU usage.

Model classification. The same procedure of model classification is used for local recognition and collaborative recognition. In local recognition, the model outputs individual activities based on features extracted from signal processing techniques and machine learning tools. For collaborative recognition, there are two types of classification models: one for queuing activity and one for queue partition. For queuing activity recognition, the support vector machine classifier does not consume much energy; for queue partition, agglomerative hierarchical clustering is adopted to divide queuers into different lines based on changing rate of relative positions of queuers.

We use Power Tutor [10] to measure the current drawn by each component in QueueSense. Basically, we keep the rest of the system and mobile phone at a nominal voltage, measuring the current when a component is active and inactive, then subtract the two and the result is the current drawn by the component. Fig. 7 shows the current of each component in QueueSense during queuing. We observe that the first major component of energy consumption is individual activity recognition. Various techniques [13], [14], [15], [16] have been developed to lower energy usage of this part, which is independent of group behavior recognition. Second to that is communication which includes propagation of individual activity features to neighbors and communication of queuer data with the remote server. This is what distinguishes our work from others. Therefore, we focus on the second component and design an energy efficiency mechanism that dynamically adjusts the communication time interval for group behavior recognition. The last part of energy consumption is queuing features extraction and classification model, which is close to 4 percent and has slight impact on mobile phone usage compared with the first two components.

4.2 Energy Efficiency Scheme

As mentioned previously, we focus on reducing energy consumption of communication for queuing behavior recognition. There are two factors impacting energy consumption on raw data sampling: networking cost and the time interval between two successive communications. Networking cost depends on specific communication methods, such as Cellular, WiFi, or Bluetooth. Detailed analysis of energy cost of different communication methods may be found in [12]. In this work, instead, we focus on carefully selecting the time interval of communication in order to reduce energy consumption for QueueSense. The larger the time interval, the more energy savings, however, the more delay in providing waiting time estimation. For instance, if we use 60 seconds as communication intervals, we process data every 60 seconds, so queue recognition would be delayed for nearly 1 minute. This communication interval is what distinguishes group behavior recognition from individual behavior recognition. Further, as pointed out by previous work [17], energy drains fast if applications keep smartphone in the awake state. In QueueSense, this issue boils down to the frequency of waking up the phone for queue recognition. One naive approach is that when customers need QueueSense, they turn it on. However, this will result in sparse queueing data, which is essentially similar to requesting people to manually label their queueing states. We, in contrast, design an adaptive approach (presented

1: **Input**: parameters of the normal distribution (μ, σ), queue line number N;
2: **Output**: interval time $T_{interval}$;
3: Initialize mobile phone:
4: $Entry = False$, $T_{interval} = \mu$;
5: **for** (*SVM.classifier*) **do**
6: classification results for queue recognition keeps stable
7: **if** ($Entry == False \bigwedge Count(N) < \lambda$) **then**
8: $Entry = True$
9: $T_{interval}$ = μ - σ
10: **else**
11: $T_{interval} += \lambda * \sigma$
12: **end if**
13: **end for**

Fig. 8. An adaptive algorithm for adjusting the interval time of communication.

below) where the interval time for queuing recognition is adjusted to accommodate dynamic service time. This scheme is a unique aspect towards energy efficient group behavior recognition that differs from previous work.

We propose a *delay-accuracy* model to characterize the impact of the interval time on the accuracy of queue waiting time estimation,

$$\rho = T_{interval}/(T_{error} + \xi), \tag{10}$$

where $T_{interval}$ is the interval time between mobile phones communicating, T_{error} is the deviation between actual and estimated waiting time, and ξ is a compensation coefficient to make sure that the denominator is not equal to zero. If the variable $T_{interval}$ is increased, the larger the variable ρ, the more energy savings.

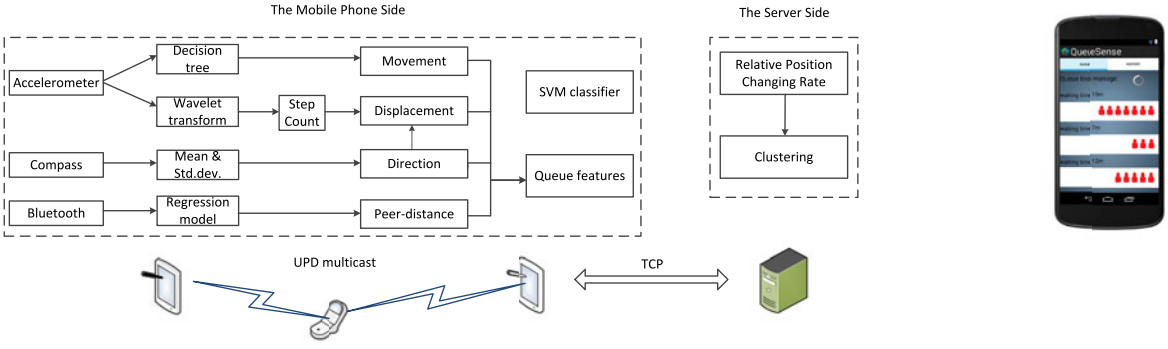
Obtaining T_{error}, however, is non-trivial. This is because that the deviation of waiting time estimation is affected by the accuracy of local and collaborative recognition on mobile phones. When T_{error} becomes large, it is unclear whether the it is caused by the increased $T_{interval}$ or the error of the classification model. We propose a benchmark for setting $T_{interval}$, i.e., the service time of queuing scenarios. Intuitively, a queuing line has a service time, so each customer keeps the previous state until the service of the previous customer is complete. Service time is used as a threshold value for the interval time between two persons waiting in lines. We set the $T_{interval}$ value around the service time so that the mobile phones can quickly reacts to changes in the queuing line. We adopt normal (or Gaussian) distribution to estimate the service time: $X \sim N(\mu, \sigma)$, where the mean and variance are trained by the historical data of the queuing scenarios. We broadcast the service time to mobile phones to preset $T_{interval}$. However, the service time changes over time. For instance, in the supermarket, the queuing service time is dependent on the cashier and the number of shopping items of customers.

We design an adaptive algorithm for adjusting $T_{interval}$ to accommodate dynamic service times. The principle is simple. Basically, when a person first enters a line, the interval time is set to be smaller than the service time. This guarantees that the queue information is responsive to newer entries. The interval time increases for people already in lines. It is because that when people are queuing, they have higher probability of queuing in the next time period. Fig. 8 presents the pseudocode for adjusting the interval time. Mobile phones receive parameters (μ, σ), N of the queuing scenarios as inputs, then go through the adjustment process before outputting $T_{interval}$. Initially, the interval time is set to be the mean value and queuing state is set *false* (Line 4). SVM classifier then detects users' queuing behavior using the approach described in the previous section (Line 6). If a user begins to queue, we set $T_{interval}$ less than μ, which aims to provide quick responses to changes in queue line information (Line 8). If the user count of the line ($Count(N)$) is low, we also set small value of $T_{interval}$ for prompt responses to changes in queue line information. If a user has been queuing for a while, we increase the $T_{interval}$ by σ (Line 11). We increase the interval time in a heuristic manner based on the intuition that queuing behavior would last until a user gets the service.

5 QueueSense Implementation

We implement a simplified prototype of QueueSense. In Fig. 9a, the individual software components in our implementation and the interactions between them are shown. On Android phones, we implement local sensing and inferring for individual activity recognition, collaborative sensing and inferring for queuing behavior recognition; and on the server, we implement queue partition, which is a part of collaborative recognition. In addition, it also provides a GUI, presenting queue information from server to users and this information includes queue length and waiting time.

1) Movement. We collect raw acceleration data at the rate of 30-50 Hz from the accelerometer, and store them in the buffer. Every acceleration frame (64 samples) is used to



(a) System architecture of QueueSense implementation (b) GUI

Fig. 9. (a) A prototype system implementation of QueueSense; (b) The queue information is shown in the GUI on a Nexus 4 Phone.

extract frequency-domain features via the FFT package. We only choose two types of movements: "still" and "moving". We assume that in most queueing scenes, people are either still or moving. We use a decision tree [7] as the classifier for this purpose. We use J.48 learning algorithm in WEKA [18] and train a tree with seven nodes and a depth of four levels. We also count the number of steps a person has walked. An entire acceleration frame is more than one second (64 samples at 30-50 Hz rate) but the frequency of people walking ranges from 0.6 to 2.5 Hz [19]. We apply discrete wavelet transform [20] to acceleration frame streaming to extract walk waveform. We can count steps by identifying the peaks of walk waveform. if time between two consecutive local peaks is people walking frequency range and the difference between the maximum and minimum is larger than $1.7g$, it is counted as a step.

2) Direction. Raw direction data is collected at the rate of 30-50 Hz from the compass. A compass frame (24 samples) is used to calculate the average value and the standard deviation. Using compass to compute direction is prone to large noise especially in indoor environments, so we suggest to combine multi-sensor to calibrate direction data as proposed in [9], [21]. We first process the compass readings to estimate the uses absolute direction. We then process the gyroscope readings to compute the angular changes to calibrate the readings from the compass, which actually are degraded by ambient magnetic field fluctuations. If the compass data does not have the similar trend to the gyroscopes readings, gyroscope data is used for calibration of compass data; otherwise the compass data is used as the direction. Also, the compass readings are periodically used as the initial bias value of the gyroscope.

3) Displacement. We use the dead reckoning technology [21] to calculate a person's moving distance. Dead reckoning is an established idea that uses the accelerometer and the compass to track a person's displacement. Basically, it multiplies the number of steps a person has walked by the step size, and then combine it with the direction of each of these steps to infer specific displacement of people in every time step.

4) Peer distance. Peer-distance is a basic metric that has been extensively applied for proximity detection. Here, we use Received Signal Strength Indicator (RSSI) of bluetooth radio to estimate peer-distance. We set up a controlled laboratory environment with four different phones at known distances and measure their RSSI. We measure the RSSI value at 70 different pre-specified distances between two phones in different environments. A regression model is constructed that depicts the relationship between RSSI level and distance. Moreover, we estimate the uncertainty error when translating RSSI to distance. There are other location methods, such as using GPS, WiFi, or audio. When queuing occurs outdoors, GPS location service [22] can be combined with Bluetooth approach. WiFi localization, when available, is suitable for indoor environments, and it requires fingerprint profiling [23]. The most accurate method is using the sound with microphone on mobile phones [24]. All these improvements can potentially help QueueSense perform better in peer distance estimation for queuing recognition.

We implement a UDP multicast client to allow mobile phones to broadcast messages to neighbors using WiFi. The exchanged message contains movement activities, direction activities, displacements and peer distances. One-hop direct communication range is sufficient to communicate with neighbors, since our queuing recognition classifier is a local classifier that only needs information from neighbors. We implement queuing feature extraction on mobile phones, including MS, DS, MCR, DCR, DD, PDCR and AD. All feature functions are handled at a fixed time interval (selection of the time interval size is discussed in Section 6). We use the SVM package to train the classifier to distinguish queuing behavior from non-queuing behavior based on features. The SVM classifier runs on the local phone. It couples with queuing features to output the result of queuing recognition.

To complement the local classifier, we implement queue partition in the central server. All queuer information is sent to the server for queue partition where a clustering algorithm in Weka [18] is used with the RPCR feature as the distance metric for clustering. We implement a TCP client to allow mobile phones to receive the queuing information from the server via WiFi. Queue line partition can be implemented in a distributed manner running on mobile phones. In a distributed implementation, when the queue length is longer than one-hop direct communication range, multihop communication is required, issues to be addressed include time synchronization and delay in obtaining information from local propagation. The distributed implementation of queue partition will make QueueSense an independent third party app running on mobile phones.

6 Performance Evaluation

In this section, we first evaluate the classification model performance of local sensing and inferring at mobile phones. We then present the detailed performance evaluation of collaborative sensing and inferring, which contains two key components in QueueSense: queuing behavior recognition and queue line partition. We report CPU and memory footprints of different components of QueueSense based on the prototype system and measure energy consumption of the system. Finally, we evaluate QueueSense in two real-world scenarios: a dining hall and a supermarket.

6.1 Local Inference Performance

In our prototype of QueueSense, we adopt four types of individual activities using local sensing and inferring on mobile phones, including *movement*, *direction*, *displacement*, and *distance*. We examine the performance of local inference based on small scale supervised experiments. We discuss accuracy in two experimental setups, a controlled experiment and an uncontrolled experiment. In the controlled experiment, we ask participants to put phones at a fixed position of their body since phone position can greatly affect the accuracy of individual activity recognition on mobile phones; in the uncontrolled experiments, we allow participants to decide where to put the phone. The results are based on 10 users who annotate their actions as the ground truth for comparison with inference outputs. The ground truth is correlated to the inference made by the local sensing and inferring on QueueSense. We carried out the scenario evaluation using leave-one-out cross-validation. Table 2 summarizes the accuracy of these four individual activity

TABLE 2
Accuracy of Local Inference

	Controlled	Normal
movement	90%	73%
displacement	85%	64%
direction	92%	77%
distance	72%	72%

recognition. For movement, the accuracy is determined by the classifier (i.e., decision tree used for movement states classification); for the other three features including direction, displacement, and distance, the accuracy is defined as the difference between the measurement inferred from sensor readings and the ground truth.

For the *movement* class, the mean accuracy is close to 90 percent in the controlled experiment and drops to 73 percent when phone position is unrestricted. For the *displacement* class, its mean accuracy remains 85 percent in the controlled experiment, but drops quickly to nearly 64 percent in the uncontrolled experiments. This is because the acceleration data of mobile phones is heavily affected by phone position. In the uncontrolled experiment, mobile phone position is a personal choice. Studies have found that phone placement (e.g., on a hand, in pants pockets) can affect the accuracy of activity inference. Furthermore, step counting is sometimes classified as errors because in our approach displacement is based on counting the different local peaks that damps the accelerometer signatures that indicate running, compared to body positions more rigidly affixed to the body. Fig. 10 shows queuers' movements over time, where each status is divided into "still" and "moving" by the decision tree classification model. A Persons' walk waveform over time under the controlled environment is shown in Fig. 11.

For the *direction* class, its mean accuracy is better in controlled experiment than uncontrolled experiments. This is because using compass data, we measure direction via the built-in calibrated API and it is easily affected by the position of smartphones. Fig. 12 shows queuers' directions over time, where each direction is measured by a compass frame.

For the *distance* class, its mean accuracy remains the same across controlled to uncontrolled experiments. For bluetooth measurement, the accuracy of the peer-distance is impacted by the environment rather than by phone context or position. We do see an impact of the environment on distance estimation accuracy, as shown in Fig. 13. Bluetooth RSSI of −10 db maps to 10 meters in the empirical regression model, and actual measurements of distance range from 12 to 8 m, we calculate the uncertainty error as 0.4 at the RSSI level of −10 db. An improvement can be made when replacing Bluetooth with a robust sensor such as acoustic sensor [24] or using sensor fusion for calibrating Bluetooth signal.

The key role of local sensing and inferring is for extracting features for recognizing queuing behavior and also dividing queuers into right lines. Since it does rely on sensors built in commodity mobile phones, its performance inevitably is affected by the sensors. In our prototype, we utilize multimodularity sensors to sense and infer individual activities, which provide basic features for extracting queuing behavior features. However, built-in sensors on mobile phones have various interference problems, which impacts the accuracy of individual activity recognition, indirectly leading to errors in queuing behavior recognition. For instance, accelerometer is sensitive to phone position on people, compass is affected by magnetic fields, and bluetooth RSSI is dependent on the surrounding environment. Calibration on acceleration data can reduce the effect of mobile phone position. Sensor fusion such as the fusion of the compass and

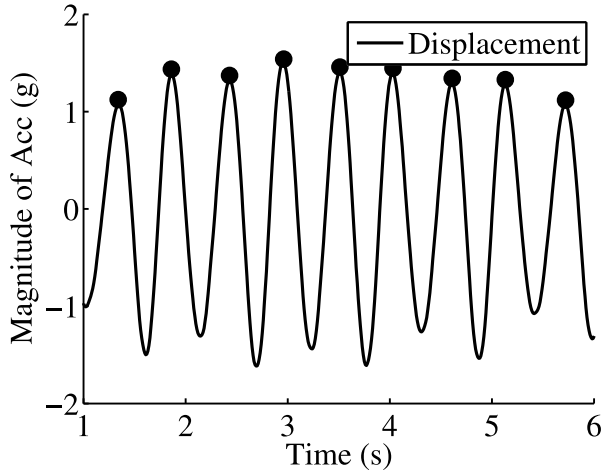


Fig. 11. A queuer's walk count over time.

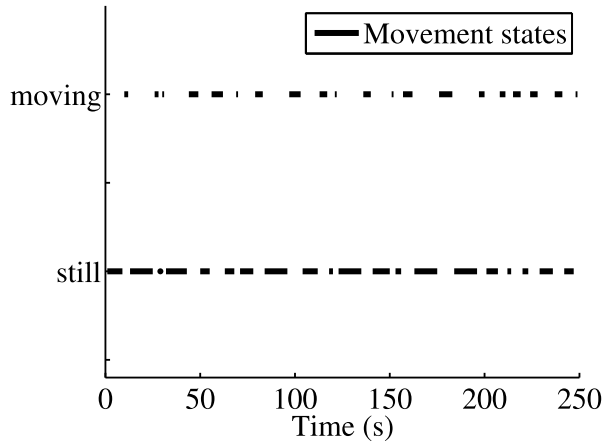


Fig. 10. A queuer's movement activities over time.

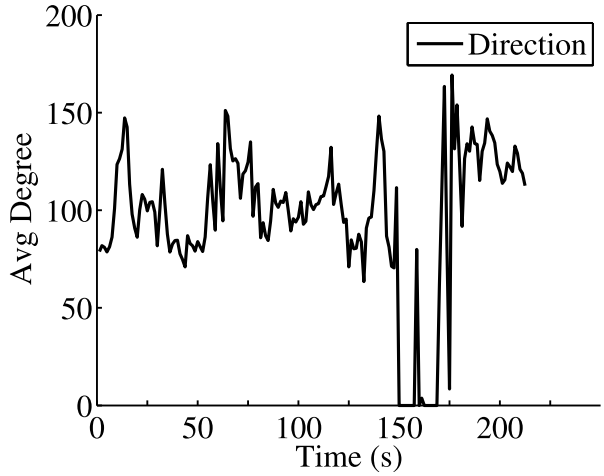


Fig. 12. A queuer's direction over time.

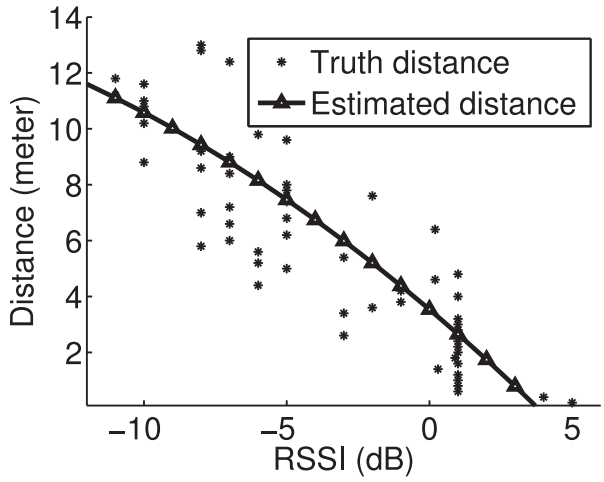


Fig. 13. Distance estimated by Bluetooth RSSI.

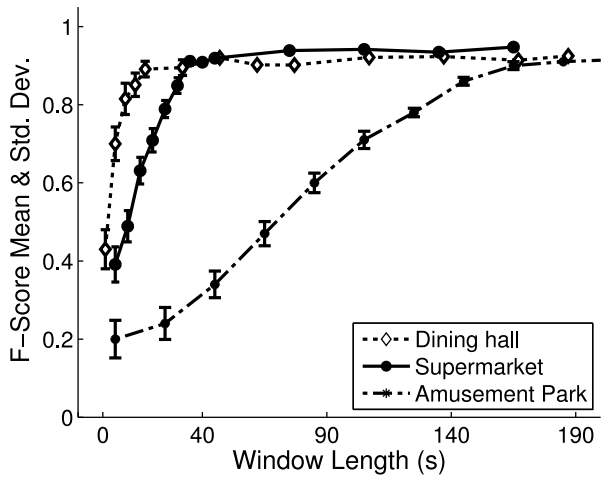


Fig. 14. Impact of time window size on accuracy of SVM classifier.

gyroscope data [21] can improve the accuracy of direction. Distance estimation can be done using other methods such as the sound with microphone on mobile phones [24]. These potential improvements will help QueueSense perform better in a variety of different queuing scenarios.

6.2 Collaborative Classification Performance

We use the data set collected from real world settings over a three month period. We conducted 40 queuing experiments in three real-life scenarios: the dining hall in the Chinese Academy of Sciences (CAS), supermarkets close to the residence hall of CAS, and the Happy Valley amusement park in Beijing. For each queue line, there are at least four people participating. During every queuing experiment, participants carried their mobile phones in a fixed position of their body, such as trousers front pocket. Participants were divided into different groups to take queuing in batches. Table 3 provides the detailed information about the experiments for collecting training data. The data recorded was synchronized and input into an offline sensor replay mechanism in MATLAB for this evaluation.

Queuing behavior recognition. We construct a SVM classifier using the training data as shown in Table 3. We are interested in how accurately people can be correctly identified as queuing. We measure the SVM classifier's accuracy using F-score as applied in Pattern recognition [7]. It reaches its best value at 1 and worst score at 0.

As mentioned previously, queuing features (i.e., MS, DS, MCR, DCR, DD, PDCR, and AD) are calculated in fixed time intervals. This time interval has a significant impact on the performance of SVM classifier's accuracy: with a small time interval, features cannot fully represent queuing characteristics; with a long time interval, delay is introduced into the classification. Fig. 14 shows the impact of window size on the accuracy of the SVM classifier in three different scenarios: a dining hall, a supermarket, and an amusement park. With the increase of the window size, the F-score is significantly increased. F-score reaches 0.9 when the windows length is close to 20 seconds in the dining hall scenario, but it needs more than 1 minute in the supermarket scenario. In the amusement park, to achieve an acceptable F-score for the SVM classifier, the window size needs to be more than 200 seconds. These results indicate that an appropriate window size should be specific to the queuing environment, such as different places, times, or seasons. We suggest that a suitable window size should be equal to the time duration between a line stopping and a line moving again. Table 4 shows precision and recall of the SVM classifier in three queuing scenarios with the window size chosen as suggested above. However, it is unnecessary to have a large training phase for all possible window lengths, because automatic recognition of queuing only needs an approximate window size. The average service time of the specific queuing scenario is a good indication of the window size and prolonging it does not improve the accuracy of queue recognition while introducing latency in providing estimated waiting time.

Queue partition. We also use F-score as the measure of queue partition accuracy. As mentioned previously, we use relative position crossing rate as the distance metric in the clustering technique for queue partition. A threshold is used to determine whether two queuers should be joined. We study the impact of different thresholds on clustering performance in three scenarios and results are shown in Fig. 15. In these experiments, we adopt the best window size shown in Table 4 to calculate RPCR values. When the RPCR threshold is between 0.3 and 0.45, the clustering in the dinning hall achieves an F-score with an average of 80 percent with a small standard deviation. While in the amusement park, the best F-score is achieved when the RPCR threshold

TABLE 3
Details of Experiments for Collecting Training Data

Scenarios	#experiments	#lines	#participants per line
Dining hall	10 times	2 or 3	8 males
Supermarket	10 times	2 or 3	6 males and 2 females
Amusement park	20 times	1	2 males and 2 females

TABLE 4
Accuracy of Queuing Behavior Recognition

Scenarios	Precision	Recall	Window Length
Dining hall	90%	92%	15 s
Supermarket	82%	86%	35 s
Amusement park	91%	71%	120 s

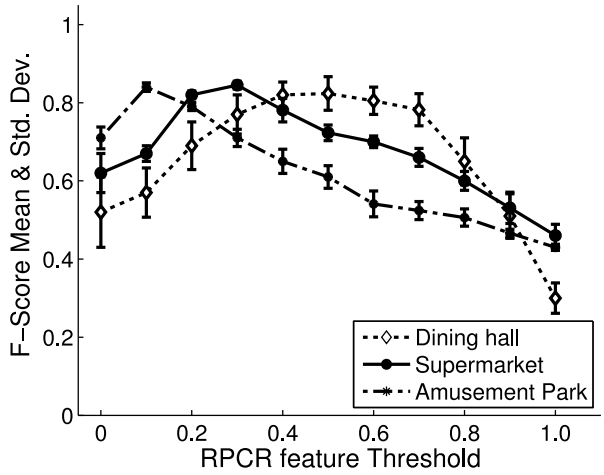


Fig. 15. Impact of RPCR threshold on accuracy of clustering.

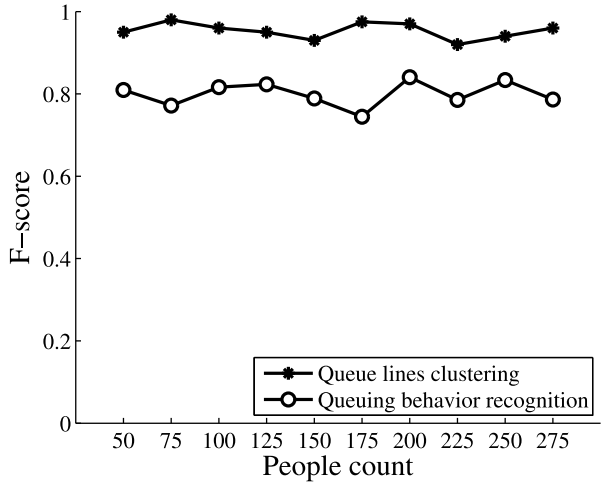


Fig. 16. Impact of people count on accuracy of QueueSense.

is close to 0.1. These results seem to indicate that the RPCR threshold needs to vary as the queuing environment changes, and needs a large training phase for each scenario. In fact, the threshold is independent of specific environment and it is the inaccurate peer distance measurement that resulted in different RPCR thresholds in different scenarios. In this prototype, we estimate peer distance using Bluetooth signal strength that is affected by the surrounding environment. Hence, three queuing scenarios have different threshold values as shown in Fig 15. If the peer-distance is measured at a finer granularity (e.g., centimeter level) using techniques such as BeepBeep [24], the threshold will not differ as much in different queuing scenarios. We believe that potential improvement in peer distance measurement will avoid the necessity to train for each scenario.

Impact of people count. Due to practical limitations, it is hard for us to recruit enough people to participate in our experiments. However, the performance of QueueSense will not degrade as people count increases. In contrast, its performance will improve. This is because group acclivities are featured by the overall characteristic of all participants, their physical activities and relationships [6]. QueueSense adopts this field theory in its design, so it will perform better if more people participate in queuing as the queuing characteristics are more distinguished. To validate this claim, we used the multi-agent simulation toolbox in Matlab to demonstrate the impact of number of people on queuing behavior recognition. We set the area size as 50 x 50 m. Every minute 10 people join the queue. There are four service counters. The entrance is the coordinate (x_0, y_0) at the bottom of the area and four service counters having coordinates (x_i, y_i) are distributed at the top of area. Each user is modeled as an agent and starts from the entrance. Every second, At each point in time, each user takes a random step away from his current position. When the time of a user staying in the area exceeds 5 minutes, with 20 percent probability that he transitions to a queuer (i.e., he moves to closest checkout counter directly). When the user moves closer to one checkout counter, the First-Come-First-Serve rule is observed, so he stands at the end of the queue. We set the queuing service time as 30 s and users are served one by one. We set optimal window size the same as service time for extracting queuing feature and RPCR feature. As shown in Fig. 16, the F-score of queuing recognition with SVM classifier is close to 80 percent, and the queue line partition has achieved nearly 95 percent. The F-scores of queuing behavior recognition in this simulation are lower than those using our data traces. The reason is that there are more randomly walking people in the simulation and that has lowered SVM model classification accuracy. Queue line partition with clustering in simulation have better performance compared with our trace data performance. This is because RPCR feature is based on peer distance measurement and we use the (x, y) coordinates to calculate their distance, which is more accurate than using Bluetooth in our prototype. In summary, performance of our queue recognition algorithm does not degrade with the increase of people count.

6.3 Resource Usage

Impact of energy efficiency mechanism on energy consumption. We first only measure the impact of the interval time $T_{interval}$ on energy consumption of QueueSense without taking into account energy consumed by other components. Then, we have performed a coarse-grained evaluation of the power consumption using empirical power models that were constructed from measurements obtained using Power Tool software [10]. CPU and memory usage is measured at last.

We compare two schemes for determining the interval time for QueueSense: one is our adaptive scheme and the other is to set the interval time to be the service time. In the service time scheme, we fix the time $T_{interval}$ as the service time. Fig. 17 is the energy consumption distribution of networking cost of the two schemes. It shows that our adaptive scheme achieves more than 25 percent energy savings compared with the service time scheme. This is because our adaptive scheme increases $T_{interval}$ as waiting time increases. The longer a user spends on waiting in line, the more energy savings our scheme gets. We further compared the cumulative distribution function (CDF) of queue recognition accuracy of the two schemes. As shown in Fig. 18, the accuracy achieved by the adaptive scheme is comparable to that achieved by the service time scheme. Combining the Figs. 17 and 18, we conclude that the adaptive scheme provides higher energy efficiency for queuing recognition without losing accuracy.

Further, we have performed experiments to evaluate the impact of running QueueSense on smartphone battery

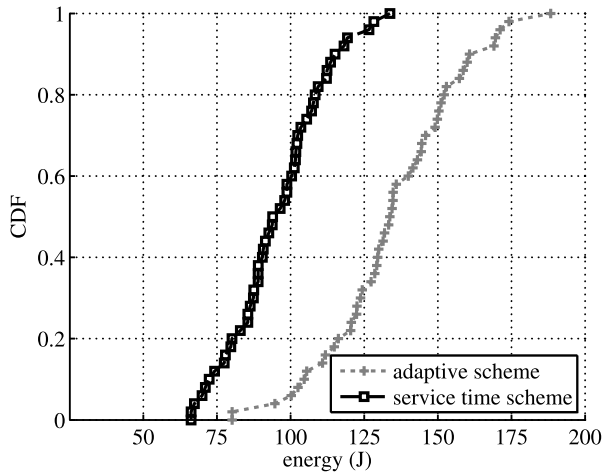


Fig. 17. Energy cost comparison of two schemes for the interval time $T_{interval}$.

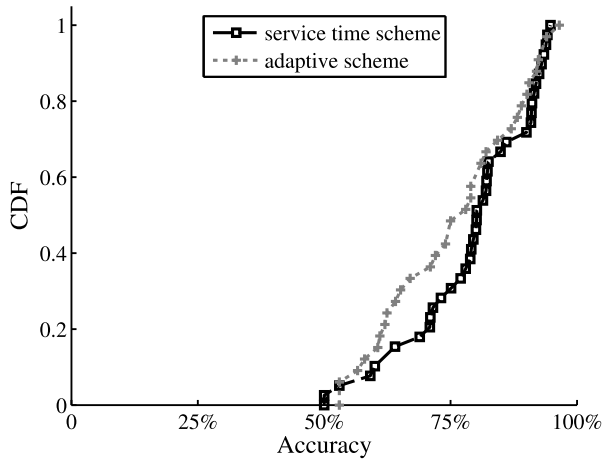


Fig. 18. Accuracy comparison of two schemes.

lifetime. As shown in Table 6, the overall energy consumption of QueueSense has a significant impact on battery lifetime. We also want to point out this battery lifetime for the QueueSensing running case is conservative here, since it let QueueSense run for the entire time. In practice, this will not be necessary as a user will only need to start QueueSense when he/she is in a queue.

CPU and memory footprints. We measure the CPU and memory footprints of QueueSense based on the prototype system with Dalvik Debug Monitor Server (DDMS). Table 5 shows the average CPU and memory usages of QueueSense. When only GUI is running, QueueSense consumes the lowest usage in CPU and memory. The CPU usage is close to 9 percent when no queuing behavior is recognized, and only reaches up to 11 percent when queuing is recognized. The memory usage is about 8.2 MB during non-queuing and slightly increases during queuing (about 8.5 MB).

TABLE 5
CPU Usage and Memory Footprints

status	*CPU usage*	*Memory usage*
GUI only	2%	3 MB
Non-queuing	9%	8.2 MB
Queuing	11%	8.5 MB

TABLE 6
Battery Duration of Mobile Phones

	Without QueueSense	*QueueSense Running*
Galaxy Nexus	35 h	21 h
Moto Milestone	19 h	9 h
HTC desire	27 h	20 h
AirWe M19	30 h	18.5 h

These results indicate that QueueSense is appropriate as a third party app on mobile phones.

6.4 Real-World Validation

We have conducted real-life experiments with 10 participants in two places: the dining hall and the Carrefour supermarket close to the residence hall of CAS. In the dining hall, we conducted 10 trials during lunch or dinner time, and the service time ranged from 10 to 60 seconds. In the supermarket, we conducted three trials, one in the morning, one in the afternoon, and one in the evening. The service time was from 20 seconds to 120 seconds. In each scenario, there was one line and the user was serviced one by one. Each user walked around in each testing location as usual. The only thing we asked all participants to do was to record the beginning and ending of their queuing behavior on their phones using the logging application we implemented. These recordings are needed for us to get the ground truth for comparison. We have set the window size (Table 4) and the RPCR threshold (Fig. 15) most appropriate for the queuing environment.

We first investigate the error in queue size estimation. We use false positive FP to denote the number of people QueueSense considers queueing when they are actually not queueing, and use false negative FN to represent the number of people QueueSense classifies as non-queuing while they are actually queueing. The error in queue size estimation is equal to the absolute difference between the two values: $|FP - FN|$. Fig. 19 shows the CDF of the error of queue size in both scenarios with ten participants queueing in each case. Sixty percent of the queue size estimation is accurate and 90 percent of the queue size estimation is off by less than three people. This result is promising.

We then estimate waiting time. The total time is equal to the number of people waiting multiplied by the service

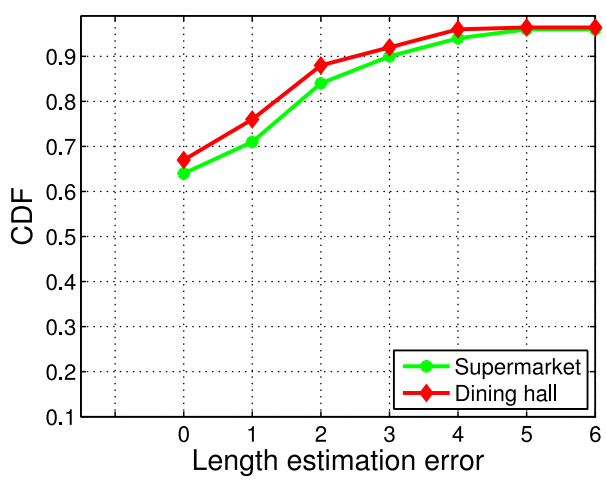


Fig. 19. Error in queue length estimation.

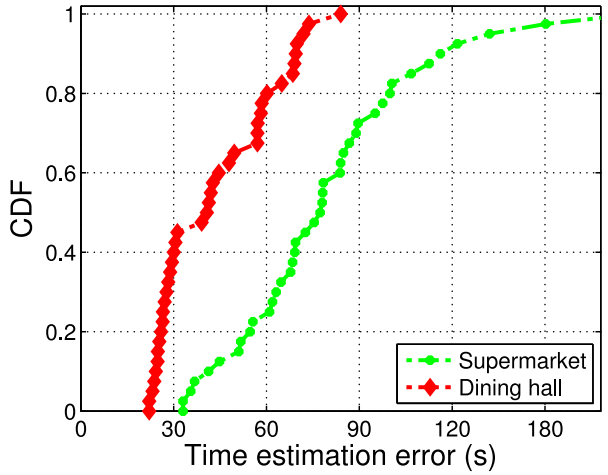


Fig. 20. Error in waiting time prediction.

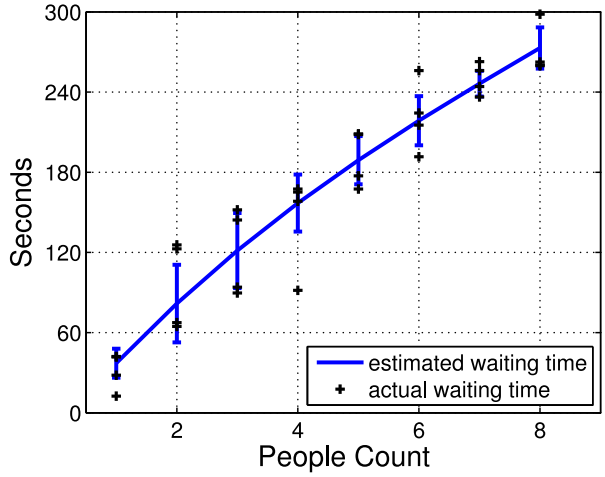


Fig. 21. Waiting time during queueing.

time. Fig. 20 plots the average error in waiting time prediction. In dining hall, nearly 80 percent of the error is below 100 seconds; in supermarket, 60 percent of the error is below 150 seconds. Although two scenarios have very similar queue sizes estimation as shown in Fig. 19, there are differences in predicting waiting time. It is because service time for each user is uncertain and purely relies on specific application scenarios. There is time deviation between estimated service time and actual service time. Furthermore, we explore the impact of people count on waiting time prediction. Fig. 21 depicts the actual waiting time deviation from the estimation when the actual number of queuers rather than estimation is used.

It is unrealistic that all customers are willing to pre-install QueueSense on their phones. When not enough people in a queue run QueueSense, the performance will drop. There are two approaches to cope with this limitation. The first approach is to let QueueSense act as a software sensor on mobile phones. When customers enter the queuing area, the queue management system sends a broadcast message to ask mobile phones to turn on this software sensor. It then continuously samples raw data just like other hardware sensors and sends results to the server. The second approach is to integrate an incentive mechanism to QueueSense to encourage the involvement of participants. A reward system can be used [25]. For instance, participants of QueueSense may be rewarded with real-time queue information. An alternative is to attract people using a game [26]. For instance, we may suggest an entertainment game to people waiting in line; when the game is being played, queuing information is collected at background and queue information is the reward. All these techniques can potentially help QueueSense perform better in coping with the issue of lacking data.

7 Related Work

There are two traditional approaches to monitor waiting lines. Video systems present an advantage as they are able to view local individual behavior and the resulting group queuing lines behavior simultaneously, and are also able to scale to larger groups. For instance, video content analysis has been used for pedestrian group detection and counting [27]. However, in these video-based approaches, occlusion where people block others may lead to detection inaccuracy. In addition, due to the limited coverage of a camera, multiple cameras are often deployed to detect the whole group in crowds. The virtual queue system is another approach to monitor queues. A customer can leave and then come back later when it is his turn, such as Disney's Fastpass [28]. However, these approaches require extra infrastructures support. In contrast, our work only relies on commodity mobile phones. Our approach can also be used to supplement these two approaches.

More recently, Bulut[29] presented LineKing, a crowdsourced line wait-time monitoring service. The approach utilized the WiFi or Location proximity to estimate queue lines. However, it is sensitive to people count. When the count of people is large or queuing space is small, the accuracy of this approach would decease quickly. In contrast, our approach relies on the similarity of individual activities, sensing and inferring by various sensors. As count of queuing people increases, the accuracy of our approach will be further improved. Similar to LineKing, Wang et al. [30] also use WiFi signals to track human queues. Our approach, instead, uses mobile phones to track human queues.

Monitoring location has also been shown to give insight into recognizing larger groups or crowds [31]. Here group behavior can be computed as a function of the location of multiple individuals and the properties of the space in which they are located. Our work differs from the previous work in that we recognize queuing groups using a combination of of widely available sensors on commodity phones including accelerometer, compass, and bluetooth. There is no need to localize each individual.

8 Conclusions

In this paper, we present QueueSense, a queuing activity recognition system by only using existing sensors on commodity mobile phones. We propose a collaborative approach for recognizing queuing behavior and design an adaptive algorithm for adjusting the communication interval time for energy efficiency on smartphones. Experimental results show that queuing behavior can be recognized with high precision and recall. In the future, in addition to address those issues mentioned in the previous section, we plan to collect more data and test QueueSense in a larger scale.

ACKNOWLEDGMENTS

L. Sun is the corresponding author. This work was supported in part by the National Natural Science Foundation of China (Grant No. 61472418) and Strategic Priority Research Program of the Chinese Academy of Sciences Grant No. XDA06040101.

REFERENCES

[1] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Commun. Mag.*, vol. 48, no. 9, pp. 140–150, Sep. 2010.
[2] K. K. Rachuri and M. E. Musolesi, "Emotionsense: A mobile phones based adaptive platform for experimental social psychology research," in *Proc. 12th ACM Int. Conf. Ubiquitous Comput.*, 2010, pp. 281–290.
[3] E. C. E. Larson, "Accurate and privacy preserving cough sensing using a low-cost microphone," in *Proc. 13th Int. Conf. Ubiquitous Comput.*, 2011, pp. 375–384.
[4] Y. Jiang and K. E. Li, "Maqs: A mobile sensing system for indoor air quality," in *Proc. Int. Conf. Ubiquitous Comput.*, 2011, pp. 493–494.
[5] J. Lester, T. Choudhury, and G. Borriello, "A practical approach to recognizing physical activities," in *Pervasive Computing*. Berlin, Germany: Springer, 2006, vol. 3968, pp. 1–16.
[6] K. Lewin and D. Cartwright, *Field Theory in Social Science: Selected Theoretical Papers*. London: U.K.: Tavistock, 1952.
[7] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006, vol. 1.
[8] R. Bennett, "Queues, customer characteristics and policies for managing waiting-lines in supermarkets," *Int. J. Retail Distrib. Manage.*, vol. 26, no. 2, pp. 78–87, 1998.
[9] I. Constandache and X. Bao, "Did you see bob?: Human localization using mobile phones," in *Proc. 16th Annu. Int. Conf. Mobile Comput. Netw.*, 2010, pp. 149–160.
[10] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. 9th Int. Conf. Mobile Syst., Appl. Serv.*, 2011, pp. 335–348.
[11] D. Chu and N. D. E. Lane, "Balancing energy, latency and accuracy for mobile sensor data classification," in *Proc. 9th ACM Conf. Embedded Netw. Sens. Syst.*, 2011, pp. 54–67.
[12] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of wifi and bluetooth in smartphones," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1363–1376, Jul. 2013.
[13] N. Roy, A. Misra, C. Julien, S. K. Das, and J. Biswas, "An energy-efficient quality adaptive framework for multi-modal sensor context recognition," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2011, pp. 63–73.
[14] S. Nath, "Ace: Exploiting correlation for energy-efficient and continuous context sensing," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 29–42.
[15] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *Proc. 8th ACM Conf. Embedded Netw. Sens. Syst.*, 2010, pp. 71–84.
[16] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proc. 8th Int. Conf. Mobile Systems, Appl., Serv.*, 2010, pp. 315–330.
[17] A. Pathak and A. E. Jindal, "What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 267–280.
[18] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Mateo, CA, USA: Morgan Kaufmann, 2005.
[19] M. Henriksen, H. Lund, R. Moe-Nilssen, and H. Bliddal, "Test–retest reliability of trunk accelerometric gait analysis," *Gait Posture*, vol. 19, no. 3, pp. 288–297, 2004.
[20] P. Barralon, N. Vuillerme, and N. Noury, "Walk detection with a kinematic sensor: Frequency and wavelet comparison," in *Proc. 28th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2006, pp. 1711–1714.
[21] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 197–210.
[22] N. Bulusu, J. Heidemann, and D. Estrin, "Gps-less low-cost outdoor localization for very small devices," *IEEE Personal Commun.*, vol. 7, no. 5, pp. 28–34, Oct. 2000.
[23] A. Haeberlen and E. E. Flannery, "Practical robust localization over large-scale 802.11 wireless networks," in *Proc. 10th Annu. Int. Conf. Mobile Comput. Netw.*, 2004, pp. 70–84.
[24] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "Beepbeep: A high accuracy acoustic ranging system using cots mobile devices," in *Proc. 5th Int. Conf. Embedded Netw. Sens. Syst.*, 2007, pp. 1–14.
[25] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 173–184.
[26] J. Deng, J. Krause, and L. Fei-Fei, "Fine-grained crowdsourcing for fine-grained recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 580–587.
[27] B. Leibe, E. Seemann, and B. Schiele, "Pedestrian detection in crowded scenes," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 878–885.
[28] Disney's fastpass. [Online]. Available: http://disneyworld.disney.go.com/guest-services/fast-pass/, 2012.
[29] M. F. E. Bulut, "Lineking: Crowdsourced line wait-time estimation using smartphones," in *Proc. Mobile Comput., Appl., Serv.*, 2013, pp. 205–224.
[30] Y. Wang, J. Yang, Y. Chen, H. Liu, M. Gruteser, and R. P. Martin, "Tracking human queues using single-point signal monitoring," in *Proc. 12th Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2014, pp. 42–54.
[31] M. B. Kjærgaard, M. Wirz, D. Roggen, and G. Tröster, "Mobile sensing of pedestrian flocks in indoor environments using WiFi signals," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2012, pp. 95–102.

Qiang Li received the BS degree in computer science from Nankai University in August 2009. Currently, he is working towards the PhD degree in the Institute of Information Engineering at the Chinese Academy of Sciences and Graduated University of Chinese Academy of Sciences. His research interests revolve around Mobile computing and smartphone applications. He would like to build prototype mobile systems based on well-founded computational models and interesting smartphone applications. He is a student member of the IEEE.

Qi Han received the PhD degree in computer science from the University of California, Irvine, in 2005. Currently, she is an associate professor in the Department of Electrical Engineering and Computer Science, Colorado School of Mines. Her research interests include distributed systems, middleware, mobile and pervasive computing, and dynamic data management, and cyber-physical systems. She is a member of the IEEE and the ACM.

Limin Sun received the BS, MS, and PhD degrees from the College of Computer, National University of Defense Technology, in China, in 1988, 1995, and 1998, respectively. Currently, he is working as a professor in the Institute of Information Engineering at the Chinese Academy of Sciences (CAS). He is an editor of the *Journal of Computer Science and Journal Computer Applications*. He is also a guest editor of special issues of *EURASIP*, *Journal of Wireless Communications and Networking*, and *Journal of Networks*. His research interests include mobile vehicle networks, delay-tolerant networks, wireless sensor networks, mobile IP, and technologies for next generation of Internet. He is a member of the IEEE.