QueueSense: Collaborative Recognition of Queuing on Mobile Phones

Qiang Li*†¶, Qi Han‡, Xiuzhen Cheng§, Limin Sun*†
* State key Laboratory of Information Security, Institute of Information Engineering, CAS, 100093
†Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS, 100093
‡Department of EECS, Colorado School of Mines, Golden, CO USA 80401
§Computer Science, The George Washington University, Washington DC, USA
¶ University of Chinese Academy of Sciences, 101408

***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 on mobile phones to assist in a queue management system. QueueSense extracts features of queuing behavior and classifies queueing via collaboration among people waiting in line. It measures the disparity of people in different lines using relative position changing rate and partitions different queues using a hierarchical clustering approach. We implement a prototype of QueueSense on Android platforms using widely available multi-modal sensors and it is the first queue detection system on mobile phones. We conduct real-world experiments at 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 with high accuracy.**

I. 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 their 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.

Mobile phones present an attractive platform for people-centric applications [1] [2] as they integrate a variety of sensors that may make similar observations as human. 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 phones. This motivates us to use commodity mobile phones to detect queuing and obtain queuing information.

978-1-4799-4657-0/14/$31.00

However, designing such a system poses two significant challenges. Firstly, queuing is a temporal group activity that is different from individual human activity recognition (HAR) [3]. A group behavior cannot be fully understood by piecemeal observations. In other words, it is infeasible to distinguish queuing from non-queuing behavior only based on individual behaviors. Secondly, multi-lines are commonly seen at public areas. Inaccurately dividing queuers into different queues results in inaccurate queue length estimation and waiting time prediction, so a fine-grained partition is necessary to classify queuers into the right lines.

We have designed QueueSense to cope with these challenges. To recognize queuing behavior, we observe that queuers have similar and relatively stable motions and directions. Hence, queuers have higher similarity and lower changing rate than non-queuers in terms of their individual temporal activities. We adopt supervised learning algorithms to build a classifier for queuing. To classify queuers into the right lines, we observe that when queuers are in the same line, their relative positions over time remain stable unless they are leaving the queue. Thus, the changing rate of relative positions among queuers in the same line is lower than queuers in different lines. We utilize unsupervised learning algorithms to partition queuers.

To validate QueueSense, we implement a prototype on Android platforms using widely available sensors such as accelerometer, compass, and bluetooth on mobile phones. We also include estimation of queue length and waiting time in the application. We run the prototype application in real-world scenarios via deploying it on the mobile phones of 10 persons who participated in waiting in lines at a dining hall and a supermarket near a university campus. Experimental results show that queuing behavior recognition has achieved high precision and recall.

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

1) We develop practical solutions to automatically recognize queuing behaviors and partition queuing lines without manual input.
2) We implement a prototype application based on QueueSense on commodity phones.

3) We conduct empirical studies of QueueSense in real world scenarios. Results show that QueueSense has achieved high accuracy.

To the best of our knowledge, QueueSense is the first work on detecting queuing using mobile phones. QueueSense has several advantages. First, it automatically detects queuing behavior without explicit inputs from users, so it does not change people's habits of mobile phone usage. Second, it only utilizes common built-in sensors without the need to modify existing mobile phone platform or infrastructure. Last, it may be used to assist in a queue manage system such as a virtual queue system or a video monitoring application.

The rest of the paper is structured as follows. We describe the related work in Section II. We then present design details on how to recognize queuing behavior and partition queues in Section III. In Section IV, we describe the implementation of a prototype application based on QueueSense. We report our real world experimental setting and empirical results to demonstrate the performance of QueueSense in Section V. Finally, we make final remarks in Section VII.

II. Related Work

There has been more and more research on smartphone-based applications [1], [2], [4]–[6]. Most previous work focuses on monitoring personal contexts such as activity, sound event, emotion, and surrounding environment. For instance, CenceMe [1] associates individual activities with social networking applications. SoundSense [2] recognizes sound event of individual surrounding in daily life. MAQS [5] monitors surrounding air quality. There is also work on detection of people emotions [4] . Instead of focusing on detection of individual activities or events, our work recognizes queueing behavior which requires collaboration of multiple persons.

Some work [7]–[9] recognizes events via collaborative phone sensing, similar to ours, but focuses on different problems. For instance, Darwin [7] use phones to collaborate and select the best quality features for improving accuracy of event detection. Detection of sound events is achieved [8] collaboratively via audio volume disparity among phones nearby. MoVi [9] uses multiple mobile phones to sense short video-clips from different times and places to form a video highlights. In contrast, our work is the first on collaboratively sensing queuing activity, which is a group behavior with interleaved activities of multiple persons.

There are two approaches to monitor waiting lines. Video systems have been used to view local queuing behavior. For instance, video content analysis has been used for pedestrian group detection and counting [10]. 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 [11]. However, these approaches requires extra infrastructures support. In contrast, our work only relies on commodity mobile phones. Our approach can also be used to supplement these two approaches.

Monitoring location has also been shown to give insight into recognizing larger groups or crowds [12]. 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.

III. QueueSense Design

Before delving into the design details of the QueueSense, we provide a formal notion of queuing. This will provide the insights necessary for distinguishing people who are queueing and who are not. We will then present an overview of QueueSense, describing the rationale behind our design. Following that discussion, we present the detailed techniques to tackle the two challenges laid out previously: queuing behavior recognition and queue partition.

A. The Notion of Queueing

Queuing is a group activity different from individual activities such as "standing", "sitting", "walking", or "running". Specifically, 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 could also occur. Therefore, it is hard to conceptually distinguish queuing from non-queuing purely based on individual activities. This also implies extracting features for queueing cannot be accomplished purely on local phones. In short, a group behavior is the characteristic of all participants rather than piecemeal properties of each individual.

In this paper, we follow Kurt Lewin's "Field Theory" [13] which states that group behavior is a function of the attributes, characteristics, and social relationships of all involved parties. Here, we provide the general notion of a queue as the function of multi-queuers according to the overall characteristics [14]: 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. 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 served (FCFS), where they have similar mobility and the sequential relationship of the mobility is stable.

This queuing notion implies that important queue features should include similarity and changing rate of the individual sequential activities. This will be used to guide our design of QueueSense.

B. QueueSense Overview

QueueSense consists of three major components as shown in Figure 1. In the following, we describe each component and the interactions between them.

Individual activity recognition. We mainly compute four features of individual activities: movement, direction, displacement, and distance. These features are used later to extract

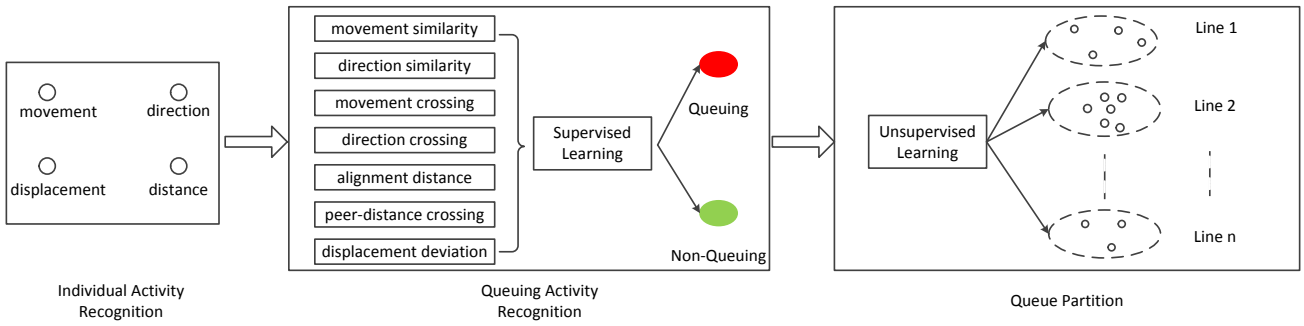


Fig. 1: Software architecture of QueueSense

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. As individual activity recognition is not the focus of this work, we defer the details of computing these four features to Section IV.

Queuing activity recognition. 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, and direction crossing rate. Second, queuers follow the rule: FCFS. Hence, we adopt peer-distance crossing rate 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 to reflect temporal relationship between queuers. QueueSense adopts a supervised learning algorithm for queuing activity recognition. Specifically, we use support vector machine to train the classification model based on the seven queuing features.

Queue partition. 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.

C. Queuing Behavior Recognition

As shown in Figure 1, 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.

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

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

$MS(a,b) = 1$ 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.

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

where

$$f(a,b)_i = 1\{avg(h_i^a) - avg(h_i^b) \leq \frac{std(h_i^a) + std(h_i^b)}{2} + G\} \tag{4}$$

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.

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 f 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.

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

where $g(a,b)_i$ 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 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 move 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.

*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 Figure 1, the model is a binary classifier that its output takes only two values, queuing or non-queuing. Considering the limitations of mobile phones, a classifier should only require lightweight processing and meanwhile can work in various real world scenarios.

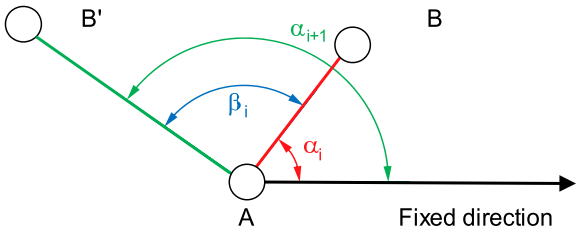


Fig. 2: The angle β_i indicates whether two persons have changed their relative position or not in two successive time slots.

We have investigated various machine learning algorithms [15] for building the queuing behavior pattern, including logistic regression, linear discriminant analysis, support vector machine (SVM), decision trees, boosting, and neural networks. We choose SVM because it is capable of generating a maximum margin classifier. It learns a hyperplane that has the largest separation or margin between queuing and non-queuing classes, so that the distance from the hyperplane is maximized. Therefore, SVM can work in various real world scenarios.

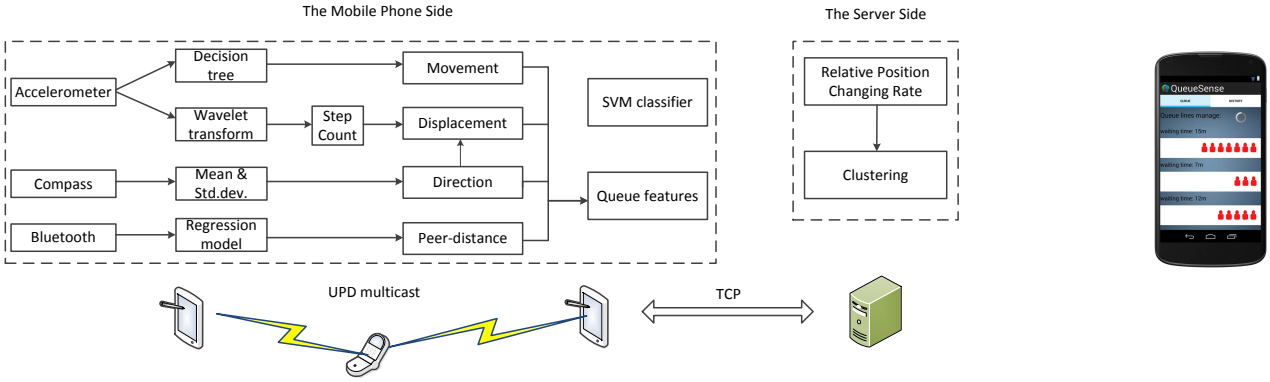
Given a data point, the queuing classifier is to decide which class the point should be in, queuing or non-queuing. The seven queuing features are viewed as a 9-dimensional vector, including MS, DS, MCR, DCR, DD, PDCR, AD×3). We adopt SVM to train a nonlinear classifier based on these features. SVM utilizes the polynomial kernel $k(x,z) = (x^T z + c)^d$ to map the original feature to a $\binom{n+d}{d}$ feature space, where n is the vector dimension and d is the polynomial order. In Section V, we describe the training process of the classifier using SVM.

D. 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.

Since the number of queues are unknown in advance, k-means clustering can not be easily applied to partition a set of queuers into lines. Instead, we apply agglomerative hierarchical clustering [15]. 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



(a) System architecture of QueueSense implementation

(b) GUI

Fig. 3: (a) A prototype system implementation of QueueSense; (b) The queue information is shown in the GUI on a Nexus 4 Phone.

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 than people who are in the same line. Therefore, We use queuers' relative position changing rate (RPCR) as the distance metric for clustering.

It is unnecessary to know relative positions for calculating their changing rate over time. Instead, we calculate RPCR using the peer distance feature, which has already been used for computing other queueing features. Let us consider the example shown in Figure 2. 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 $RPCR$ 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.

IV. QueueSense Implementation

We implement a simplified prototype of QueueSense (Figure 3a). On Android phones, we implement individual activity recognition and queuing behavior recognition; and on the server, we implement queue partition. In addition, it also provides a GUI as Figure 3b shown, presenting queue information from server to users and this information includes queue length and waiting time. Our current version is approximately 2,300 lines of code using JAVA.

Queuing features are derived from four individual activity features including movement, direction, displacement, and peer distance. These individual features are computed using multi-modality sensors already built in commodity mobile phones: accelerometer, compass, and bluetooth.

1) Movement. Using the standard Android SDK, 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 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 [15] as the classifier for this purpose. We use J.48 learning algorithm in WEKA [16] and train a tree with 7 nodes and a depth of 4 levels. Figure 4 shows two queuers' movements over time, where each status is divided into "still" and "moving" by the decision tree. 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 Hz to 2.5 Hz [17]. We apply discrete wavelet transform [18] to acceleration frame streaming to extract walk waveform. Figure 5 shows two persons' walk waveform over time. 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 different 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. Figure 6 shows two queuers' directions over time, where each direction is measured by a compass frame.

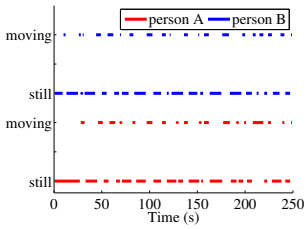


Fig. 4: Two queuers' movement activities over time.

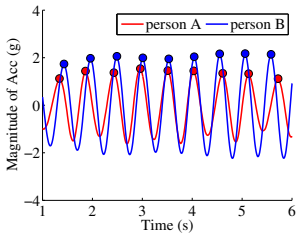


Fig. 5: Two queuers' walk count over time.

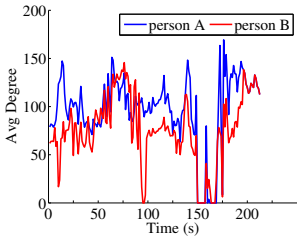


Fig. 6: Two queuers' direction over time.

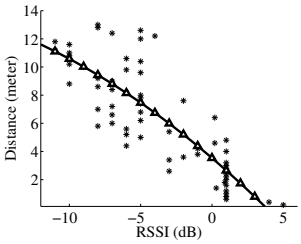


Fig. 7: Distance estimated by Bluetooth RSSI.

3) Displacement. We use the dead reckoning technology [19] 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. There are various approaches to measure relative distance between peers, such as Bluetooth, WiFi or Audio. 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 measured the RSSI value at 70 different prior-set distances between two phones in different environments. A regression model is constructed that depicts the relationship between RSSI level and distance. Moreover, we estimated the uncertainty error when translating RSSI to distance. As shown in Figure 7, bluetooth RSSI of -10db maps to 10meters in the empirical regression model, and actual measurements of distance range from 12m to 8m, we calculate the uncertainty error as 0.4 at the RSSI level of -10db.

We implement a UDP multicast client to allow mobile phones to broadcast messages to neighbors. The exchanged message contains movement activities, direction activities, displacements and peer distances. Here, we assume that one-hop direct communication range is sufficient to communicate with neighbors. We also implement a TCP client to allow mobile phones to receive the queuing information from the server.

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 V). We use the SVM package to train the classifier to distinguish queuing behavior from non-queuing behavior based on features. The SVM classifier is running on the local phone. It couples with queuing features to output the result of queuing behavior. We use the clustering algorithm in Weka [16] to implement clustering in the server. We replace the distance metric in clustering with the RPCR feature.

TABLE I: CPU usage and memory footprints.

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

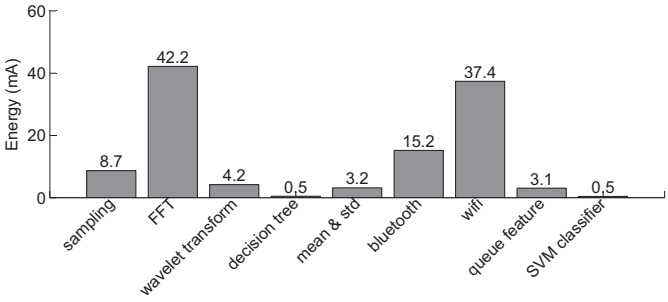


Fig. 8: The current drawn by each component in the application based on QueueSense.

V. Performance Evaluation

In this section, we discuss the evaluation of QueueSense. We first measure CPU and memory footprints, and energy consumptions of different components of QueueSense based on the prototype system. Following this, we present the detailed performance evaluation of the two key components of QueueSense: queuing behavior recognition and queue line partition. Finally, we evaluate QueueSense in two real-world scenarios: a dining hall and a supermarket.

A. System Measurements

Resource Usage. We measure the CPU and memory footprints of QueueSense based on the prototype system with a GUI part. We use Dalvik Debug Monitor Server (DDMS) coupled with Android platform to measure the CPU and memory usages. Table I 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% when no queuing behavior is recognized, and only reaches up to 11% when queuing is recognized. The memory usage is about 8.2 MB during non-queuing and slightly increases during queuing (about 8.5MB). These results

TABLE II: Details of experiments for collecting training data

Scenarios	Number of experiments	Participants
Dining hall	10 times	8 males
Supermarket	10 times	6 males and 2 females
Amusement park	20 times	2 males and 2 females

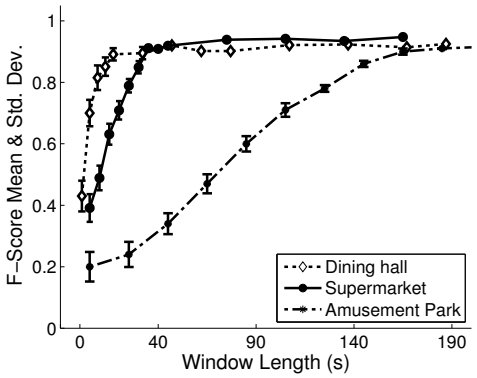


Fig. 9: Impact of time window size on accuracy of SVM classifier

TABLE III: Accuracy of queuing behavior recognition

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

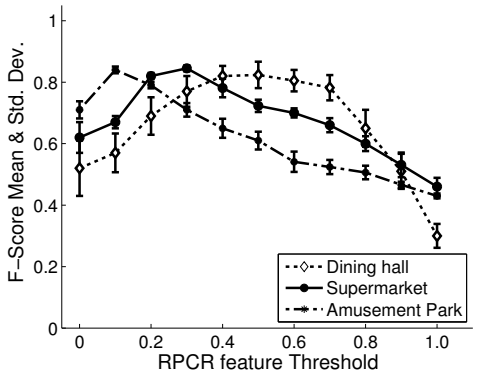


Fig. 10: Impact of RPCR threshold on accuracy of clustering.

indicate that QueueSense is appropriate as a third party app on mobile phones. There is a slight increase in resource usage when queuing compared with non-queuing. This is because when queuing is identified, an extra step is taken for further queue partition.

Energy Measurements. We use Power Tool software [20] to measure the current drawn by each component in the prototype 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. Figure 8 shows the current of each component in QueueSense during queuing. We observe that the most energy consuming component is individual activity recognition. Second to that is communication which includes propagation of individual activity features to neighbors and communication of queuer data with the remote server. The current drawn by features extraction and classification of queuing behavior is only a small part of the energy consumption in total, close to 4%. These results indicate that the energy consumption of QueueSense has slight impact on mobile phone usage if we only focus on features extraction and queuing recognition.

B. Classification Performance

We separately evaluate the performance of the SVM classifier for queuing behavior recognition and the hierarchical clustering for queue partition. 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. During every queuing experiment, participants carried their mobile phones in fixed position of their body, such as trousers front pocket. A logging service continuously recorded raw sensor data in the SQLite database supported by Android SDK. Participants were divided into different groups to take queuing in batches. Table II 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 II. 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 [15]. The F-score can be interpreted as a weighted average of the precision and recall:

$$Fscore = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

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 short time interval, features cannot fully represent queuing characteristics; with a long time interval, delay is introduced into the classification. We study the tradeoff between time window length and the performance of the SVM classifier. Figure 9 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

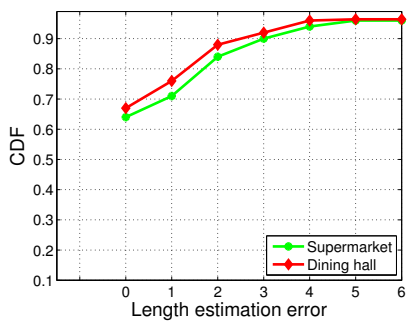


Fig. 11: Error in queue length estimation

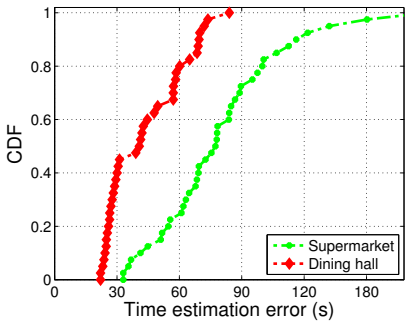


Fig. 12: Error in waiting time prediction

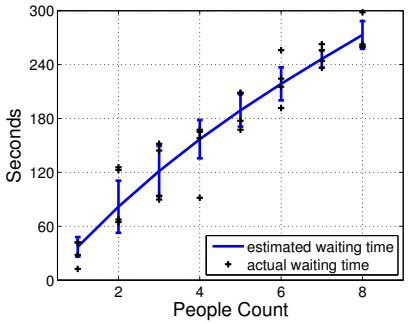


Fig. 13: Waiting time during queueing

needs to be more than 200 seconds. These results indicate that an appropriate window size should be specific to the queuing environment. We suggest that the suitable window size should be equal to the time duration between a line stopping and a line moving again. Table III shows precision and recall of the SVM classifier in three queuing scenarios with the window size chosen as suggested above.

Queue Partition. We also use F-score as the measure of queue partition accuracy. As mentioned previously, we use relative position crossing rate (RPCR) 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 Figure 10. In these experiments, we adopt the best window size shown in Table III to calculate RPCR values. When the RPCR threshold is between 0.3 to 0.45, the clustering in the dinning hall achieves an F-score with an average of 80% with a small standard deviation. While in the amusement park, the best F-score is achieved when the RPCR threshold is close to 0.1. These results indicate that the RPCR threshold needs to vary as the queuing environment changes. The reason is that QueueSense uses peer distance and displacement to calculate RPCR. In this prototype, we adopt Bluetooth signal strength to measure peer distance. Bluetooth signal is effected by the surrounding environment.

C. Queue Information

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. In the supermarket, we conducted 3 trials, one in the morning, one in the afternoon, and one in the evening. The mobile phone used by every participant is different including Google NexusS, Nexus4, Motorola milestone, HTC desire, and AirWe M19. 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 III) and the RPCR threshold (Figure 10) 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: $\mid FP - FN \mid$. Figure 11 shows the CDF of the error of queue size in both scenarios with ten participants queueing in each case. 60% of the queue size estimation is accurate and 90% of the queue size estimation is off by less than 3 people. This result is promising.

We then estimate waiting time on QueueSense. The total time is equal to the number of people multiplying by the service time. We adopt a joint Gaussian distribution to estimate the service time, where the distribution is based on historical service time pluses with white noise. Figure 12 plots the average error in waiting time prediction. In dining hall, nearly 80% of the error is below 100 seconds; in supermarket, 60% of the error is below 150 seconds. Although two scenarios have very similar queue sizes estimation as shown in 11, 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. Figure 13 depicts the actual waiting time deviation from the estimation when the actual number of queuers rather than estimation is used.

VI. Discussion

Sensor Interference: The key contribution of QuequeSense is its ability of 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 multi-modularity 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 gyroscope data [19] can improve the accuracy of direction. Distance estimation can be done using other methods such as the sound with microphone on mobile phones [21]. These potential improvements will help QueueSense perform better in a variety of different queuing scenarios.

QueueSense Installation: We assume that predicting the waiting time and queue length requires that most queuers' mobile phones have installed QueueSense. It is unrealistic that all customers are willing to pre-install QueueSense on their phones. There are two approaches to cope with this problem. The first approach is that QueueSense acts as a software sensor as the built-in sensor on mobile phones. When customers enter into the queuing area, the queue management system send a broadcast message to ask mobile phones to turn on this software sensor. It is continually sampling raw data as like hardware sensors and sends results to the server. Instead, queuing behavior recognition and queue partition will be done at the central server. The second one is that QueueSense integrates an incentive mechanism to encourage the involvement of participators.

Distributed Approach: QueueSense recognizes queuing behavior in a distributed manner, but partitions queuers into different lines in a centralized manner. The clustering for queue partition can be modified to a distributed implementation on mobile phones. However, it will raise the issue that queue length might be longer than one-hop direct communication range. Multi-hop communication is require to consider time synchronization and delay of obtain globe information from local propagation. This improvement will make QueueSense an independent third party app running on mobile phones.

VII. Conclusions

In this paper, we present QueueSense, a queuing activity recognition system by only using existing sensors on commodity mobile phones. We use seven queuing features and a SVM classifier to distinguish between queuing and non-queuing. We apply hierarchical clustering to divide queuers into right lines. To validate QueueSense, we implement a prototype application on Android platform and run it in real-world environments. Experimental results show that queuing behavior can be recognized with high precision and recall.

Acknowledgment

Limin Sun is corresponding author. This work was supported in part by the National Natural Science Foundation of China (Grant No.61202066), and National High Technology Research and Development Program of China (Grant No.2013AA014002).

References

[1] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *SenSys '08*, 2008, pp. 337–350.

[2] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Soundsense: scalable sound sensing for people-centric applications on mobile phones," in *MobiSys '09*, 2009, pp. 165–178.

[3] J. Lester, T. Choudhury, and G. Borriello, "A practical approach to recognizing physical activities," in *Pervasive Computing*. Springer Berlin / Heidelberg, 2006, vol. 3968, pp. 1–16.

[4] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "Emotionsense: a mobile phones based adaptive platform for experimental social psychology research," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ser. Ubicomp '10, 2010, pp. 281–290.

[5] Y. Jiang, K. Li, L. Tian, R. Piedrahita, X. Yun, O. Mansata, Q. Lv, R. P. Dick, M. Hannigan, and L. Shang, "Maqs: a mobile sensing system for indoor air quality," in *Proceedings of the 13th international conference on Ubiquitous computing*. New York, NY, USA: ACM, 2011, pp. 493–494.

[6] M. Azizyan, I. Constandache, and R. Roy Choudhury, "Surroundsense: mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th annual international conference on Mobile computing and networking*. ACM, 2009, pp. 261–272.

[7] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin phones: the evolution of sensing and inference on mobile phones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 5–20.

[8] Y. Lee, C. Min, C. Hwang, J. Lee, I. Hwang, Y. Ju, C. Yoo, M. Moon, U. Lee, and J. Song, "Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion," in *Mobisys'13*, 2013.

[9] X. Bao and R. Roy Choudhury, "Movi: mobile phone based video highlights via collaborative sensing," in *8th Mobisys*, ser. MobiSys '10, 2010, pp. 357–370.

[10] B. Leibe, E. Seemann, and B. Schiele, "Pedestrian detection in crowded scenes," in *CVPR 2005.*, vol. 1. IEEE, 2005, pp. 878–885.

[11] Disney's fastpass. [Online]. Available: http://disneyworld.disney.go.com/guest-services/fast-pass/

[12] M. B. Kjærgaard, M. Wirz, D. Roggen, and G. Tröster, "Mobile Sensing of Pedestrian Flocks in Indoor Environments using WiFi Signals," in *Proceedings of the 10th IEEE International Conference on Pervasive Computing and Communications (PerCom 2012)*, 2012.

[13] K. Lewin and D. Cartwright, *Field theory in social science: Selected theoretical papers*. Tavistock London, 1952.

[14] R. Bennett, "Queues, customer characteristics and policies for managing waiting-lines in supermarkets," *International Journal of Retail & Distribution Management*, vol. 26, no. 2, pp. 78–87, 1998.

[15] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. springer New York, 2006, vol. 1.

[16] I. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[17] 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.

[18] P. Barralon, N. Vuillerme, and N. Noury, "Walk detection with a kinematic sensor: Frequency and wavelet comparison," in *EMBS'06.* IEEE, 2006, pp. 1711–1714.

[19] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *10th Mobisys*. ACM, 2012, pp. 197–210.

[20] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *9th Mobisys*. ACM, 2011, pp. 335–348.

[21] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "Beepbeep: a high accuracy acoustic ranging system using cots mobile devices," in *5th Sensys*. ACM, 2007, pp. 1–14.