Qnalyzer: Queuing Recognition Using Accelerometer and Wi-Fi Signals

Zone-Ze Wu, Cheng-Wei Wu, Lan-Da Van, Yu-Chee Tseng*
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
E-mail: {wuzz, cww0403, ldvan, yctseng*}@cs.nctu.edu.tw

***Abstract*—*Queuing recognition* is a recently new raised research topic, which uses sensors of smartphones to automatically recognize human queuing behaviors. However, existing collaborative approaches need to exchange sensor data among nearby smartphones, causing extra communication overheads and even delay. In view of this, this work proposes a new framework called *Qnalyzer* for *queuing recognition using accelerometer and Wi-Fi signals*. It consists of three tiers. The first tier is run by each individual smartphone to identify each user's context without exchanging data with nearby smartphones. A new algorithm called *QCF* (*Queuer and non-queuer ClassiFier*) is proposed, which considers mixture features of accelerometer and Wi-Fi signals to effectively identify whether the user is queuing or not. The second tier is an algorithm called *QCT* (*Queuers ClusTering*) running at the server side to effectively identify which queuers belong to which queues based on users' movement features. The third tier is an estimation model called *QPE* (*Queue Property Estimation*) for measuring waiting time, service time, and queue lengths. The Qnalyzer prototype on Android smartphones and the corresponding performance evaluations under real-life queuing scenarios are implemented. The extensive experiment results show that Qnalyzer achieves good performance with high accuracy.**

Keywords*—*behavior sensing, group activity recognition, machine learning, queuing, mobile computing.

I. INTRODUCTION

Queuing is a common *group activity* frequently happening in our daily life at restaurants, movie theaters, and supermarkets. However, when lining up in a queue, people will be eager to know the waiting time and service time of the queue. When there are multiple queues to choose, knowing their properties beforehand is certainly very helpful. Providing such information is beneficial to both customers and business managers. For customers, this information can help them to choose a service line having the shortest waiting time for participation, which can save their valuable time and increase their loyalty. For managers, such information may help them to better allocate resources.

Smartphone has become an essential tool for most people. In addition to telecommunication, it is embedded with a variety of sensors, such as accelerometer, gyroscope, proximity and magnetic sensors. *Queuing recognition* by smartphones [1][17][19][21] is a recently new raised research topic, which uses sensors of smartphones to automatically recognize human queuing behaviors. Some approaches have been proposed for this new research direction. *LineKing* [1] exploits Wi-Fi signal strengths and GPS locations to estimate waiting time of queuers. Wag *et al.* [20] considered Wi-Fi signals to distinguish a queuer's queuing status, including waiting, being served, and leaving. However, these two approaches are designed for *single-line queuing* scenarios. *Multiple-line queuing* scenarios are studied in *QueueSense* [19] and *QueueVadis* [17]. The former takes a *collaborative* approach, while the latter takes a *non-collaborative* one. The collaborative approach identifies whether a user is queuing or not by considering the relative behaviors of nearby users. This requires message exchange among nearby smartphones and a lot of calculations. Therefore, *QueueSense* may incur extra communication overheads and even delays in finding queuing status. On the contrary, *QueueVadis* identifies whether a user is queuing or not by his/her individual activities, thus, avoiding the inter-phone communications.

Although *QueueSense* and *QueueVadis* have considered the multiple-queues scenarios, they did not consider the sensing data from Wi-Fi (Wireless Fidelity) [9][13][20]. Wi-Fi is a widely used wireless communication technology that allows electronic devices to connect with Internet. Owning to the merits of low-cost installation, wide-range coverage and fast data transfer, more and more places have provided free Wi-Fi. As indicated in [20], putting a Wi-Fi monitor near the head of a queue (e.g., service desk) allows us to use Wi-Fi signals to approximately estimate the distance between a queuer and the service desk. Such information is very useful for queuing recognition. Our goal is to propose a new framework for queuing recognition using such information.

The queuing scenario considered in this work is sketched in Fig. 1. The targeted field contains both queuers and non-queuers. Queuers may form up to K ($K \geq 1$) queues, where each queue is nearly straight. These N service counters are on the same side and there is a Wi-Fi AP placed at the center of these counters. Each user is assumed to carry a smartphone in his/her pocket. Under the considered queuing scenario, the three important research questions are:

- Queuer identification: how to effectively recognize whether users are queuing or not?
- Queuers separation: how to correctly separate queuers into the right queue and identify their orders in queues?
- Queue property estimation: how to accurately estimate queue properties, like waiting time and service time, for different queues?

Answering the above three questions positively is not easy. Developing a queuing recognition system using accelerometer and Wi-Fi signals will pose several challenges.

First, to make machines be able to automatically and effectively recognize human queuing behaviors, we need to

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

analyze behaviors of queuers and non-queuers, and explore queuing patterns hidden in sensing data to develop algorithms. If such patterns cannot be found and nicely integrated into the system, the system may fail to automatic recognition.

Second, to recognize queuing behaviors, a promising solution is to adapt machine learning methods. However, to construct a robust recognition model, we need to address several issues, including data pre-processing, feature extraction, model construction, and online recognition. Every step is crucial and has significant impact on performance of the resulting model.

Third, although a collaborative approach requires inter-phones communications, it can obtain more information from nearby users for queuer identification. However, only individual's information can be used by a non-collaborative approach for queuer identification.

In this paper, we address all of the above challenges by proposing a new framework called *Qnalyzer* for *queuing recognition using accelerometer and Wi-Fi signals*. We integrate queuing patterns hidden in accelerometer and Wi-Fi data, and incorporate them into Qnalyzer. The major contributions of this work are summarized below:

First, we propose an novel algorithm called *QCF* (*Queuer and Non-queuer Classification*), which adopts supervised machine learning methods for effective queuer identification. It first extracts movement and Wi-Fi features from accelerometer and Wi-Fi signal strength of individual's smartphone, and then nicely integrates them to build an interpretable *decision tree* [23] for effective recognition of queuers and non-queuers. In the experiments, the accuracy of QCF is up to 10% better than the QueueVadis [17].

Second, a novel algorithm named *QCT* (*Queuers Clustering*) is proposed, which adapts unsupervised machine learning methods to effectively cluster queuers into the right queues. Because queuers in the same queue have similar sequential mobility, it uses the movement features extracted by QCF to cluster queuers. Then, it adopts a regression method to effectively infer orders of queuers in the queues based on the Wi-Fi features of queuers.

Third, we propose a new method called *QPE* (*Queue Property Estimation*) to effectively estimate queue properties of queues based on the recognition results of QCT, where the estimated properties include queue lengths, waiting time and service time. It differs from previous estimation methods [17][19] in that it allows to achieve better estimation based on recent queuing behaviors of users. The experiments show that the estimation error of service time of QPE is less than 5 seconds.

Finally, we implemented the Qnalyzer prototype and tested its performance under different real-life queuing scenarios (two restaurants and a McDonald). Empirical experimental results show that *Qnalyzer* not only achieves high accuracy but also performs better than the current best non-collaborative method [17].

The rest of this paper is organized as follows. Related works are briefly reviewed in Section II. Section III introduces the details of the proposed framework and algorithms. Empirical experiments conducted on real-life scenarios are presented in Section V. Finally, conclusion and future works are presented in Section VI.

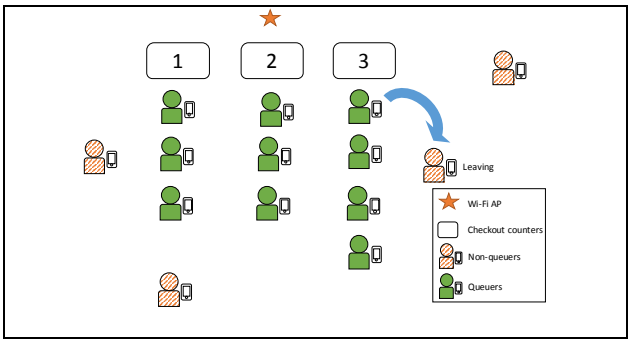

Fig. 1. The queuing scenario considered in this work.

II. Related Work

In this section, we survey studies related to *queuing recognition*, including *vision-based approaches*, *Wi-Fi-based approaches*, and *smartphone-based approaches*.

(1) Vision-based Approaches. Some studies [5][12][15] relying on surveillance camers have been proposed to recognize pedestrian groups. Based on the image processing technologies, they can detect movement direction of crows and count the number of people in crows. However, they could not automatically identify people who are queuers and who are not in a crowd. Besides, they usually require to deploy multiple cameras at multiple public areas, which may involve high installation costs and privacy issues. Moreover, the quality of recognized results can be easily degraded by factors like lighting conditions, camera positions, and background interference.

(2) Wi-Fi-based Approaches. *LineKing* [1] is a crowdsensing waiting time estimation system, which utilizes network localization and a Wi-Fi AP to coarsely detect and estimate waiting time of queuers in a queue. A new framework [20] based on single point Wi-Fi monitoring is proposed to infer queuing status of queuers in a queue, including queuing, be served, and leaving. Although the above two studies exploit the nice capability of Wi-Fi for waiting time estimation, they are designed for the single queue scenario with the assumption that all the users are in a queue. In other words, they did not address the challenging problems of queuer recognition and queuers separation in multiple queues scenarios.

(3) Smartphone-based Approaches. *QTime* proposed in [21] is a smartphone-based approach for queuing time notification, but it did not take the problem of queuer sepration into account. *QueueSense* [19] and *QueueVadis* [17] consider the problem of queuing recognition in multiple queues scenarios using smartphones. QueueSense recognizes queing behaviors of users based on collaborative exchanges of the individual sensing data with neighboring smartphones, which may involve privacy issue and high computational cost. QueueVadis adopts a two-layer classifier to recognize whether the user is queuing or not by indivisual sensing data, and identifies whether two queuers are in the same queue by a same-queue classifier. However, QueueSense and QueueVadis are designed based on accelerometer and compass, which did not consider the Wi-Fi signals.

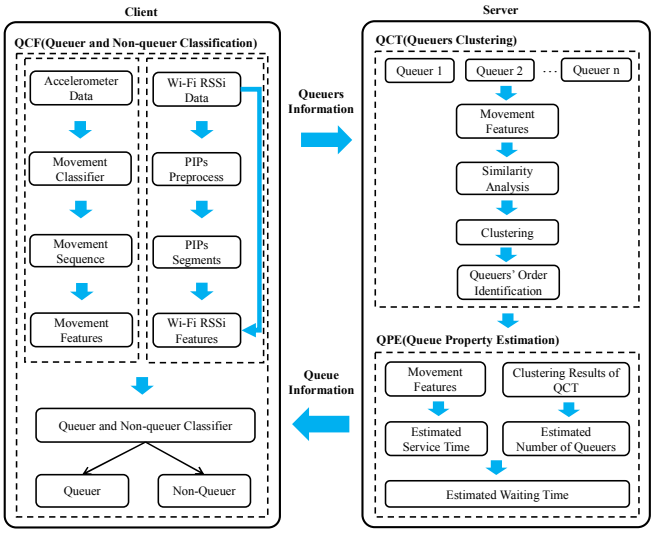

Fig. 2. System architecture of the proposed framework.

$$\langle S\,S\,S\,S\,S\,S\,S\,S\,S\,W\,W\,S\,S\,S\,S\,S\,S\,S\,S\,W\,S\,S\,S\,S\rangle$$

Fig. 3. Illustration of a movement sequence of a user, where notations *W* and *S* represent *walking* and *standing* activities respectively.

III. The Qnalyzer System

The system architecture of Qnalyzer is shown in Fig. 2. It is a client-server architecture consisting of three modules: (1) *QCF* (*Queuer and non-queuer ClassiFier*), (2) *QCT* (*Queuers ClusTering*) and (3) *QPE* (*Queue Property Estimation*). QCF is performed at the client side, while QCT and QPE are performed at the server side. The whole system is run in real-time fashion and the details of each module are described as follows.

A. The QCF Algorithm

As shown in left side of Fig. 2, QCF consists of three main stages: (1) *movement feature extraction*, (2) *Wi-Fi feature extraction*, and (3) *queuer and non-queuer classification*.

(1) Movement Feature Extraction

Queuing is the behavior where users comply with the first-come and first-served rule. If a queuer in the front of a queue steps forward, then the other queuers behind he/she will step forward later. Comparing with non-queuers, queuers have regular intermittent movement interleaved with moving forward (i.e., *walking*) and stop-to-wait (i.e., *standing*), while non-queuers' movement behaviors are less regular. Based on this queuing pattern, QCF extracts movement features from accelerometer in the user's smartphone to reflect his/her sequential movement behaviors.

The raw accelerometer data is collected at the sampling rate of 50 Hz. It consists of three time series along *x*-axis, *y*-axis, and *z*-axis, representing accelerations in front-and-back, up-and-down, left-and-right directions, respectively. A non-overlapped sliding window is adopted to segment accelerometer data [6]. Each window consists of ω consecutive segments, where the length of each segment is θ seconds. Based on our internal tests, θ and ω are set to 1 and 24, respectively.

Fig. 4. Illustration of special queuing patterns hidden in the RSSI trace.

For each axis of the accelerometer data in the segment, we extract low-level features *mean*, *variance*, *zero crossing rate*. Besides, we extract low-level features *covariance* and *correlation* for each pair of axis of the accelerometer data in the segment.

After low-level feature extraction, QCF adopts a pre-established classifier, called *movement classifier*, to classify each segment. The classifier takes extracted features as inputs and outputs per segment a *movement activity* (i.e., *walking* or *standing*). This classifier is constructed from 150 labeled instances via *J48 decision tree* algorithm in *Weka* [23]. The ω identified movement activities captured by each window is also called *the user's movement sequence*, representing the user's sequential moving behavior within ω seconds.

For example, Fig. 3 shows a user's movement sequence, where notations *W* and *S* represent *walking* and *standing* activities, respectively. Let *A* be either walking or standing activities. For each movement sequence *M*, we extract the following high-level movement features from *M*.

- ***Chg*:** the times that two adjacent movement activities change from *walking* to *standing* or vice versa.
- ***Num*(*A*):** the number of movement activities that are *A* (i.e., walking or standing).
- ***Min*(*A*):** the minimum number of consecutive occurrences of *A*.
- ***Max*(*A*):** the maximum number of consecutive occurrences of *A*.
- ***Avg*(*A*):** the average number of consecutive occurrences of *A*.

(2) Wi-Fi Feature Extraction

Recall the considered queuing scenario depicted in Fig. 1, a Wi-Fi AP is put at the center of checkout counters. It periodically transmits packets to users' smartphones. Therefore, users' smartphones can measure the Wi-Fi RSSI (Received Signal Strength Indicator) from the AP per second. The strength of Wi-Fi RSSI indirectly reflects the approximate distance between a user and a service counter. As shown in Fig. 4, a Wi-Fi RSSI trace of a queuer exhibits three unique queuing patterns. First, when the queuer gradually moves forward the checkout counter, the RSSI will increase gradually. Second, when the queuer is served at the checkout counter, the RSSI will be very strong and stable. Third, when the queuer completes the transaction and leaves from the checkout counter, the RSSI will decrease dramatically.

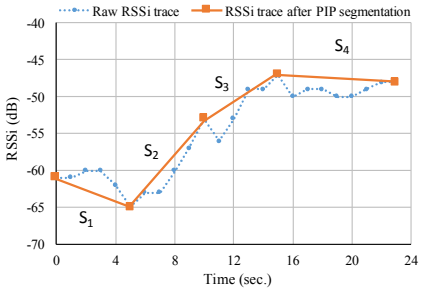

Fig. 5. A RSSI trace segmented by the PIP segmentation with $\kappa = 5$.

To capture the above unique queuing patterns, QCF extracts *strength-based features* and *trend-based features* of Wi-Fi RSSI trace captured by the window. Let R be the RSSI trace captured by the window WD. We extract the following three strength-based Wi-Fi features from R.

- ***AvgR*:** the average of RSSI values.
- ***RangeR*:** the range of RSSI values.
- ***StableR*:** the number of RSSI values that are greater than a user-specified threshold δ. In our test, δ is set to -40.

For trend-based Wi-Fi feature extraction, QCF first adopts the *PIP* (*Perceptual Important Point*) algorithm [22] to segment the RSSI trace, and then extracts six trend-based Wi-Fi features from the segmented RSSI trace. Given a user-specified parameter κ, the RSSI trace will be segmented by κ ($2 \le \kappa \le \omega$) critical points. In our settings, κ is set to 5. Fig. 5 shows an example of a real RSSI trace segmented by the PIP segmentation with $\kappa = 5$. Let G be the set of segments generated by adopting PIP algorithm on R. Then, QCF extracts the following six trend-based Wi-Fi features from G.

- ***NPS*:** the number of positive slopes in G.
- ***NNS*:** the number of negative slopes in G.
- ***MaxS*:** the maximum slope in G.
- ***MinS*:** the minimum slope in G.
- ***MCPS*:** the maximum duration of the consecutive positive slopes in G.
- ***MCNS*:** the maximum duration of the consecutive negative slopes in G.

(3) Queuer and Non-queuer Classification

After extracting the movement and Wi-Fi features, QCF adopts a pre-established classifier, called *queuer and non-queuer classifier*, to classify each window. The classifier takes extracted features as inputs and outputs per window a class label to indicate whether the user is queuing or non-queuing. This classifier is constructed from training data collected in advance via *J48 decision tree* algorithm in *Weka* [23].

B. The QCT Algorithm

Previous approach [17] considers compass data for queuers separation. However, it may not perform well in our considered scenario, where each queue is nearly straight and every queuer faces nearly the same direction. Thus, the information of compass data could not be used to effectively distinguish which queuers belong to which queue. Intuitively, queuers in the same queues have similar sequential mobility, while queuers in the different queues have dissimilar movement behaviors.

TABLE I. AN EXAMPLE OF DISTANCE MATRIX

	Chg	*Num(S)*	*Num(W)*	*Min(S)*	*Min(W)*	*Max(S)*	*Max(W)*	*Avg(S)*	*Avg(W)*
Queuer A	3	19	5	2	2	18	3	9.5	2.5
Queuer B	3	20	4	2	2	19	2	10	2
Queuer C	4	15	9	7	2	8	4	7.5	3
Queuer D	4	14	10	6	3	8	4	7	3.33

For instance, in a queue, when the queuer in the front of a queue steps forward, then the other queuers behind he/she will step forward too. As comparing with queuers in different queues, queuers in the same queues have more similar movement behaviors and stationary durations. Based on this queuing pattern, we propose the QCT algorithm. The right top of Fig. 2 shows the corresponding main workflow.

The main purpose of QCT is to aggregate all the queuers' data at the server side to cluster queuers into the right queues. The movement features are reused to calculate the similarity between queuers. Then, it adopts an unsupervised machine learning method to cluster queuers into groups. For example, Table I shows a data matrix consisting of nine movement features of four queuers. In Table I, each queuer can be regarded as a data point in a nine dimensional coordinates. QCT calculates the distance between every two data points to represent their similarity. If the distance between two data points is smaller, the similarity between them is higher. That means they have higher chance to be in the same queue. To measure the similarity of queuers, QCT first adopts *z-score normalization* [11] to normalize data, and then calculates distance between every two queuers by *standardized Euclidean distance*. After similarity calculation, QCF then adopts an agglomerative hierarchical clustering algorithm [19] to cluster queuers. To compute the distance between two clusters, QCT adopts the *average linkage* [11].

C. The QPE Algorithm

The workflow of QPE is shown in the right bottom of Fig. 2. QPE uses movement features extracted by QCF and clustering results of QCT to estimate the following queue properties: (*i*) *queue lengths*, (*ii*) *the service time for each queue*, and (*iii*) *the waiting time for each queue*.

Let $CR = \langle Q_1, Q_2, \ldots, Q_K \rangle$ be the clustering result of QCF, where Q_i ($1 \le i \le K$) is the i-th cluster in CR. Let $Q = \langle q_1, q_2, \ldots, q_m \rangle$ be an arbitrary cluster in CR, where it consists of m ($m \ge 1$) queuers identified by QCF and q_j ($1 \le j \le m$) represents the j-th queuer in Q. The *queue length* of Q estimated by QPE is the number of queuers in Q. In a queue, queuers behind the first queuers step forward when the first queuer completes the service and leaves the checkout counter. We observe that the service time is very close to the maximum duration of *standing* activity in a window. Therefore, we average the maximum duration of standing activity of each queuer in the same cluster as the service time for the cluster Q. The service time for a cluster Q estimated by QPE is denoted as $AvgST(Q)$. Notice that, unlike previous approach [19] which assumes that the service time is a constant for all the cashier lines, the service time estimated by QPE is not a constant. It will dynamically adjust service time based on the recent behaviors of queuers. The waiting time for a cluster Q estimated by QPE is defined as $AvgST(Q) \times |Q|$, that is, the multiplication of estimated service time for Q and the number of queuers in Q.

IV. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed algorithms and compare them with QueueVadis [17], a state-of-the-art approach for non-collaborative queuing recognition. Experiments of QCF were performed on each participant's smartphone (Google Nexus 4), which is equipped with a 1.5 GHz Qualcomm S4 APQ8064 Processor and 2 GB of memory, running on Android 4.2. The results of QCF are sent to the server. Data in server are stored in the MYSQL database. Experiments of QCT and QPE were performed on a server, which is equipped with a 2.4 GHz Intel® Core™2 Quad Processor Q6600 and 4 GB memory, running on Windows 7. All the algorithms are implemented in Java.

A. Performance Evaluation of QCF

First, we evaluate the performance of movement classifier in QCF. To construct movement classifier, we use a smartphone to collect training data and testing data. The raw accelerometer is collected at the sampling rate of 50Hz. Table II shows charateristics of the datasets. In Table II, the notations #Walking and #Standing represent the number of instances labeled by *Walking* and *Standing*.

TABLE II. TRAINING AND TESTING DATASETS FOR MOVEMENT CLASSIFIER

Dataset	Type	#Walking	#Standing	Instances
MS_TrainDS	Training Data	75	75	150
MS_TestDS	Testing Data	50	50	100

We evaluate the performance of the classifier by metrics *precision*, *recall*, *F-Measure*, and *accuracy*. Let $C = \langle c_1, c_2, ..., c_n \rangle$ be the set of classes in the training dataset, for each class c_i ($1 \le i \le n$), we can construct a confusion matrix for c_i [11] based on the classification results on a testing dataset, as shown in Table III.

Table IV shows a confusion matrix based on the testing dataset in Table II. In Table IV, the notations *TP, FP, FN* and *TN* represent numbers of *true positives*, *false positive, false negatives* and *true positives* [11]. Based on the confusion matrix, the definitions of *precision*, *recall*, *F-Measure*, and *accuracy* are shown below:

- $Precision(c_i) = \frac{TP}{TP + FP}$
- $Recall(c_i) = \frac{TP}{TP + FN}$
- $F\text{-}Measure = \frac{2 \times Precision(c_i) \times Recall(c_i)}{Precision(c_i) + Recall(c_i)}$
- $Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$

Table IV shows the confusion matrix for the classification result of the constructed movement classifier on the MS_TestDS dataset. As shown in Table IV, the classifier has a very high accuracy, which is up to 97%. Table V shows the precision, recall and F-Measure for each class. As shown in Table V, the classifier has very good performance to distingush different movement activities (i.e., *walking* and *standing*). The experiments show that the constructed movement classifier servers as a good basis for consequent processes of Qnalyzer.

TABLE III. AN EXAMPLE OF CONFUSION MATRIX

	Classified as c_i	Classified as not c_i
Ground truth is c_i	*TP*	*FN*
Ground truth is not c_i	*FP*	*TN*

TABLE IV. CONFUSION MATRIX FOR THE MOVEMENT CLASSIFIER

	Classified as Walking	Classified as Standing
Actual Walking	48	2
Actual Standing	1	49

TABLE V. PERFORMANCE OF MOVEMENT CLASSIFIER

Class	Precision	Recall	F-Measure
Walking	96.1%	98%	97%
Standing	98%	94%	94%

TABLE VI. TRAINING DATASETS FOR QUEUER AND NON-QUEUER CLASSIFIERS

Dataset	Scenario	#Exp.	Type	#QR	#NQ
FR_TrainDS	First Restaurant	25	1 queue	50	50
SR_TrainDS	Second Restaurant	20	2 queues	50	50
MC_TrainDS	McDonalds	20	3 queues	50	50

TABLE VII. TESTING DATASETS FOR EVALUATING QNALYZER

Dataset	Scenario	#Exp.	Type	#QR	#NQ
FR_TestDS	First Restaurant	25	1 queue	50	50
SR_TestDS	Second Restaurant	20	2 queues	50	50
MC_TestDS	McDonalds	20	3 queues	50	50

Then, we evaluate the performance of the queuer and non-queuer classifier in QCF. We collect training and testing datasets from three different places: First Restaurant, Second Restaurant, and McDonalds. The three training datasets are denoted as FR_TrainDS, SR_TrainDS, and MC_TrainDS, respectively. The three testing datasets are denoted as FR_TestDS, SR_TestDS, and MC_TestDS, respectively. Tables VI and VII show the data characteristics of the datasets. In these tables, the notations #QR and #NQ represent the numbers of instances labeled as *queuers* and *non-queuers*, respectively. We use the metrics *accuracy*, *F-Measure*, *macro-average precision* and *macro-average recall* to evaluate the effectiveness of the classifiers. Based on Table III, the macro-average precision and recall are defined as

- $Macro\text{-}average\ Precision = \frac{1}{n}\sum_{i=1}^{n} Precision(c_i)$.
- $Macro\text{-}average\ Recall = \frac{1}{n}\sum_{i=1}^{n} Recall(c_i)$

Fig. 6 shows the experimental results of the classifiers. As shown in Fig. 6, the classifier has good discriminative ability for recognizing queuers and non-queuers. For example, the accuracy of QCF on FR_TestDS, SR_TestDS, and MC_TestDS are 85%, 82% and 80%, respectively. This is because that the integration of movement and Wi-Fi features makes QCF be able to achieve such good performance. Besides, QCF also has good performance in term of F-Measure. For example, in Fig. 6, the F-Measure of QCF on SR_TestDS is up to 85%. Next, we compare the accuracy of QCF and QueueVadis [17]. We implement QueueVadis in Java. As shown in Fig. 7, the accuracy of QCF is higher than that of QueueVadis about 10%. This is because that QueueVadis only uses the features of accelerometer without considering that of Wi-Fi signals. On the contrary, QCF considers not only movement features but also Wi-Fi features for queuer identification. Thus, it can achieve more accurate recognition performance than QueueVadis.

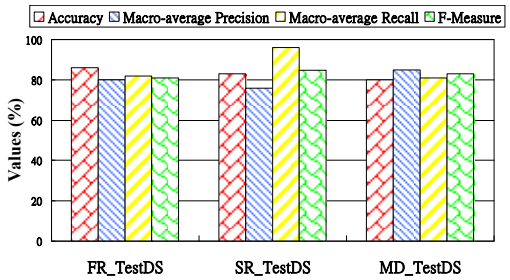

Fig. 6. The performance of the queuer and non-queuer classifiers in QCF.

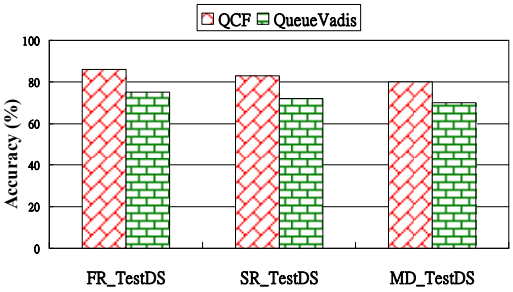

Fig. 7. The accuracy of queuer identification for QCF and QueueVadis.

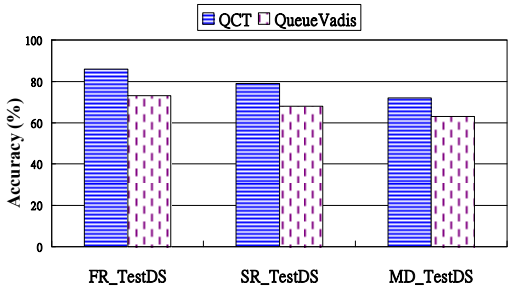

Fig. 8. The accuracy of queuers seperation for QCT and QueueVadis.

B. Performance Evaluation of QCT

In this subsection, we compare the accuracy of QCT with QueueVadis. The distance thresholds used in QCT on FR_TsetDS, SR_TestDS and MD_TestDS are set to 3.2, 3.5, 4.2, respectively. Based on the results of queuers separation on a testing dataset, a confusion matrix can be constructed as Table VIII.

TABLE VIII. AN EXAMPLE OF CONFUSION MATRIX FOR QUEUERS SEPARATION

	Recognized into same queue	Recognized into different queue
Ground truth is The same queue	*TP*	*FN*
Ground truth is different queue	*FP*	*TN*

If two queuers are in the same queue in actual, and they are recognized as being in the same queue by the method, then this case will be counted as a true positive. If two queuers are not in the same queue in actual, and they are recognized as being not in the same queue by the method, then this case will be counted as a true negative. The accuracy of queuers separation is defined as

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}.$$

Fig. 8 shows the accuracy of QCT and QueueVadis on the three testing datasets. As shown in Fig. 8, the accuracy of QCT is much higher than that of QueueVadis, where the performance improvement is over 10%. For example, the accuracies of QCT on SR_TestDS is up to 79%, while that of QueueVadis is only 68%. The reason why QCT performs much better than QueueVadis is because that QCT uses better results from QCF as inputs for queuers separation, while QueueVadis uses the low-quality results from its two-layer classifiers as inputs for queuers separation. Another reason is that, in our considered scenarios, queues are nearly straight. Therefore, the direction information of queuers used in QueueVadis will degrade its accuracy for queuers separation. However, QCT did not use such information, and it only uses movement features of queuers to separate queuers via agglomerative hierarchical clustering. Therefore, Qnalyzer can achieve much better accuracy than QueueVadis.

C. Performance Evaluation of QPE

In this subsection, we evaluate the performance of QPE. Fig. 9 shows the averages of actual service time and the estimated service times on the three testing datasets. As shown in Fig. 9, the error of the estimation results of QPE on FR_TestDS and SR_TestDS are less than 2 seconds. Besides, on MC_TestDS, the error of the estimation results of QPE is less than 5 seconds. Fig. 10 shows the averages of actual waiting time and the estimated waiting times on the three testing datasets. As shown in Fig. 10, the error of the estimation results of QPE is about 16 seconds. Then, we evaluate the *error rate* and *root mean square error* (*RMSE*) of the estimation results of QPE, including queue lengths, service times, and waiting times. Let $AV = \langle av_1, av_2, ..., av_n \rangle$ be the set of ground truth values, and $EV = \langle ev_1, ev_2, ..., ev_n \rangle$ be the set of estimated values, the *error rate* is defined as

$$ER(AV, EV) = \sum_{i=1}^{n} \frac{|av_i - ev_i|}{av_i},$$

and RMSE is defined as

$$RMSE(AV, EV) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(av_i - ev_i)^2}.$$

Table IX shows the error rates and RMSEs of queue lengths, service time, and waiting times. In Table IX, the abbreviations QL, ST, and WT represent queue length, service time, and waiting time, respectively. As shown in Table IX, QPE has low estimation error rates and RMSEs. This is because QPE estimates service time and waiting time of queuers based on recent behavior of users rather than a fixed constant. This allows QPE achieves better estimation results.

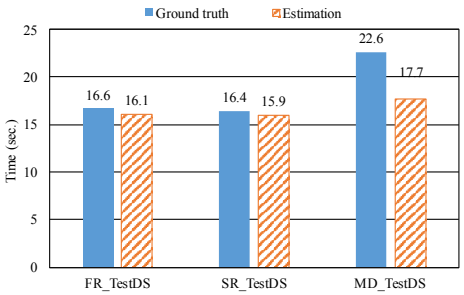

Fig. 9. The averages of actual service time and the estimated service time.

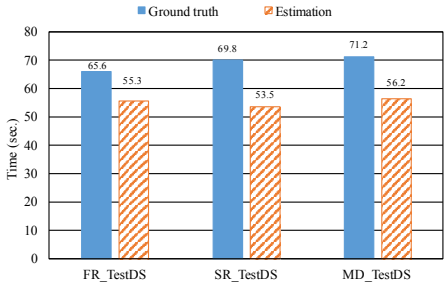

Fig. 10. The averages of actual waiting time and the estimated waiting time.

TABLE IX. THE ERROR RATES AND RMSES FOR QUEUE LENGTH, WAITING TTIME AND SERVICE TIME

	Error Rate			RMSE		
Dataset	QL	ST	WT	QL	ST	WT
FR_TestDS	0.24	0.14	0.26	0.98	3.68	12.05
SR_TestDS	0.25	0.14	0.25	1.00	3.92	10.87
MC_TestDS	0.38	0.16	0.36	1.25	6.58	13.01

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel framework called *Qnalyzer* for *effective queuing recognition using accelerometer and Wi-Fi signals*, which has never been explored so far. First, we propose a new algorithm called *QCF*, which considers mixture features of accelerometer and Wi-Fi signals for effective queuers recognition. Second, we propose a novel algorithm named *QCT*, which relies on users' movement features and hierarchical clustering to separate queuers into the right queues, and identifies queuers' orders in queues by Wi-Fi features. Finally, a self-adaptive estimation method called *QPE* is proposed to accurately estimate waiting times, service times, and queue lengths. We have conducted real-life experiments to evaluate Qnalyzer. In the experiments, the accuracies of Qnalyzer for queuer identification and queuers separation are better than that of QueueVadis [17] about 10%. Besides, the error of average service time estimated Qnalyzer is less than 5 seconds. Results show that Qnalyzer servers as an effective solution to the new problem of queuing recognition using accelerometer and Wi-Fi signals. In the near future, the heading different directions in the queue such as snake queue will be our research target.

REFERENCES

[1] F. Bulut, M. Demirbas, and H. Ferhatosmanoglu. "LineKing: coffee shop wait-time monitoring using smartphones," *IEEE Transactions on Mobile Computing*, vol. 14, no. 10, pp. 2045-2058, 2015.

[2] M. Beigl, H. Gellersen, and A. Schmidt. "Mediacups: experience with design and use of computer-augmented everyday artefacts," *Computer. Network*, vol. 35, no. 4, pp.401-409, 2001.

[3] C. Bishop and N. Nasrabadi. "Pattern recognition and machine learning," *Spring Science Business Media*, vol. 128, 2006.

[4] F. Bernhard and H. Riedwyl. "Standard distance in univariate and multivariate analysis," *The American Statistician*, vol.40, no.3, pp.249-251, 1986.

[5] M. Chang, N. Krahnstoever, S. Lim, and T. Yu. "Group level activity recognition in crowded environments across multiple cameras," in *Proc. of Advanced Video and Signal Based Surveillance*, pp. 56-63, 2010.

[6] B. Chun and P. Maniatis. "Augmented smartphone applications through clone cloud execution," in *Proc. of the 15th Workshop on Hot Topics in Operating Systems*, vol. 9 , pp. 8-11, 2009.

[7] C. Florence. "Multiple sequence alignment with hierarchical clustering," *Nucleic acids research*, vol.16, no. 22, pp. 10881-10890, 1988.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. "Diversity in smartphone usage," in *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pp. 179-194, 2010.

[9] C. Gayathri, V. Tam, V. Alexander, G. Marco. M. Richard, Y. Jie, and Y. Chen. "Vehicular speed estimation using received signal strength from mobile phones," in *Proc. of the 12th ACM International Conference on Ubiquitous Computing*, pp. 237-240, 2010.

[10] T. Huynh, M. Fritz, and B. Schiele. "Discover of activity patterns using topic models," in *Proc. of the 2008 Ubiquitous Computing*, pp. 10-19, 2008.

[11] J. Han, M. Kamber, and J. Pei. "Data mining: concepts and techniques, 3rd edition," *Morgan Kaufmann*, 2011.

[12] T. Joshua, D. Vin, and L. John. "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319-2323, 2000

[13] J. Manweiler, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, "Predicting length of stay at wifi hotspots," in *Proc. of the IEEE International Conference on Computer Communications*, pp. 3102-3110, 2013.

[14] 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 *Proc. of the 7th International Conference on Mobile Systems*, pp. 165-178, 2009.

[15] B. Leibe, E. Seemann, and B. Schiele, "Pedestrian detection in crowded scenes," in *Proc. of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recogntion*, vol. 1, pp. 878-885, 2005.

[16] 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 *Proc. of the 6th ACM Conference on Embedded Network Sensor System*, pp. 337-350, 2008.

[17] T. Okoshi, Y. Lu, C. Vig, Y. Lee, R. Balan, and A. Misra. "QueueVadis: Queuing analytics using smartphones," in *Proc. of the 14th International Conference on Information Processing in Sensor Networks*, pp. 214-225, 2015.

[18] Y. Park and J. Chen. "Acceptance and adoption of the innovative use of smartphone," *Industrial Management and Data Systems*, vol. 107, no. 9, pp. 1349-1365, 2007.

[19] L. Qiang, Q. Han, X. Chengand, and L. Sun. "Collaborative recognition of queuing behavior on mobile phones," *IEEE Transactions on Mobile Computing*, vol. 15, no. 1, pp. 60-73, 2016.

[20] Y. Wang, J. Yang, Y. Chen, H. Liu, M. Gruteser, and P. Martin. "Tracking human queues using single-point signal monitoring," in *Proc. of the 12th Annual International Conference on Mobile Systems, Applications and Services*, pp. 42-54 2014.

[21] Y. Wang, J. Wang, and X. Zhang. "QTime: a queuing-time notification system based on participatory sensing data," in *Proc. of the 37th Annual Computer Software and Applications Conference*, pp.770-777, 2013.

[22] Z. Zhe, J. Jiang, and H. Wang. "A new segmentation algorithm to stock time series based in PIP approach," in *Proc. of the 2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 5609-5612, 2007.

[23] Weka 3: data mining software in Java,
Avaiable at: http://www.cs.waikato.ac.nz/ml/weka/.