LineKing: Coffee Shop Wait-Time Monitoring Using Smartphones

Muhammed Fatih Bulut, *Member, IEEE,* Murat Demirbas, *Member, IEEE,* and Hakan Ferhatosmanoglu, *Member, IEEE*

Abstract—This article describes LineKing, a crowdsensing system for monitoring and forecasting coffee shop line wait times. LineKing consists of a smartphone component that provides automatic and accurate wait-time detection, and a cloud backend that uses the collected data to provide accurate wait-time estimation. LineKing is used on a daily basis by hundreds of users to monitor the wait-times of a coffee shop in the University at Buffalo, SUNY. The novel wait-time estimation algorithms of LineKing deployed at the cloud backend provide median absolute errors of less than 3 minutes.

Index Terms—Crowdsensing, Smartphone applications, Wait-time estimation

1 Introduction

Long and unpredictable lines at coffee shops are inconveniences of city life. A webservice that provides real-time monitoring and estimation of line wait-time would help us make informed choices and improve the quality of our lives. Furthermore, understanding line waiting has benefits beyond improving the end-user experience because this has been a long standing problem in the operations research area.

Traditional solutions to the line wait-time monitoring are based on infrastructure-based solutions such as camera placement, sensor deployment or monitoring signals that are generated through bluetooth or Wi-Fi capable devices [1], [2], [3], [4], [5]. However, these solutions usually do not scale well as they are costly to deploy and mostly designed for specific places. In order for the line wait-time monitoring service to become widely adopted the service should be infrastructure-free, fairly-accurate, easy-to-use, and should work automatically without relying on manual input from users. In this article, we take a novel approach and try to solve the line wait-time monitoring problem through crowdsensing with smartphones.

Addressing line wait-time monitoring problem using smartphones has two main challenges. First, we need to detect the wait-time using off-the-shelf sensors that are provided by the smartphones. However, wait-time detection using smartphones requires use of costly location sensors to understand the presence of the user at the coffee shop. Moreover, to understand if the user is waiting in the line, or sitting at the coffee shop, we may need to use additional sensors, such as accelerometer, to recognize the relevant activity. But, performing all of these operations in smartphones may drain user's battery very quickly and therefore has the risk of being deemed unattractive for users to install our application.

Second, since this is a crowdsensing architecture, we may not always have a person waiting in the line. We find that even when our automated wait-time detection component is returning dozens of readings daily, these readings are still too sparse and non-uniform to provide accurate answers to real-time queries about line wait-time. Additionally, newly arriving customers may not experience the same wait-time as the one who leaves the line. Hence as a second challenge, we need to estimate the current and future wait-time using the sparse and non-uniform previous history of the collected data.

Our method to address the wait-time detection is to utilize the low-cost network location provider of Android. We calculate the distance of the user from the coffee shop to dynamically set the location-sensing frequency. Since the network localization provides coarse-grained information, once we make sure that the user is around the coffee shop, we periodically scan the Wireless Access Points (WAP) around the user. By exploiting the unique fingerprint of WAP beacons in the coffee shop, we detect the entrance and exit in high precision. For the coffee shop we performed our experiments, majority of the customers pick their orders to-go, and therefore after appropriately filtering out the outliers, we obtain the wait-time of the customers. Moreover, to scale LineKing to other coffee shops and franchises, we propose an improvement to the wait-time detection component further by utilizing the activity recognition techniques using the accelerometer sensor on the smartphones.

Our method to address the second challenge, the wait-time estimation problem, is based on a search in the previous history of the collected data. More specifically, to overcome the difficulty of constructing time-series data from sparse and non-uniform data, our solution in essence finds the best k candidates from the past data in order to estimate the future wait-time.

Our contributions are as follows:

1) We designed, implemented, and deployed a crowdsourced line wait-time estimation system called Line-

- *M. F. Bulut and M. Demirbas are both with the Computer Science and Engineering Department of University at Buffalo, SUNY. E-mail: {mbulut, demirbas}@buffalo.edu.*
- *Hakan Ferhatosmanoglu is with the Computer Engineering Department of Bilkent University. E-mail: hakan@cs.bilkent.edu.tr.*

King (LK). Our LK app [1] for Android platform has been downloaded by more than 1000 users in our university, and are used on a daily basis by hundreds of users to monitor the wait-time of a coffee shop in the student union of our university. To the best of our knowledge, LK is the first automated crowdsensing wait-time estimation service.

2) As a part of LK, we implemented a fully automatic and accurate wait-time detection component on Android platform. Our method utilizes network localization and WAP scanning capabilities of Android to detect presence of the user at the coffee shop.
3) As a part of the wait-time detection component, we developed a lightweight, variance-based activity recognition unit to detect the wait-time more accurately by utilizing continuous streams of accelerometer data. **Our experiments showed that, we can detect the actual wait-time of a user with a median error of 20 seconds accuracy.**
4) Our solution to wait-time estimation problem works well with non-uniform and sparse data by finding the best possible k candidates using the regression analysis on the previous history of wait-times. **Our results indicated that LK can estimate the wait-time of the coffee shop with less than 3 min. median absolute error.** We believe that, our solution for wait-time estimation problem can be extended to other similar crowdsensing systems which have sparse and non-uniform data.

Outline of the rest of the paper. We describe the model and assumptions of our deployment next. Section 3 presents the wait-time detection component of LK. In Section 4, we explain how we improve the wait-time detection component by using activity recognition. In Section 5, we discuss LK's wait-time estimation component along with the experimental results. Section 6 discusses how to scale LK to other coffee shops and franchises. Section 7 discusses the challenges we faced during the development and deployment process of LK. Finally, we present the related work in Section 8 and conclude with Section 9.

2 Model and Assumptions

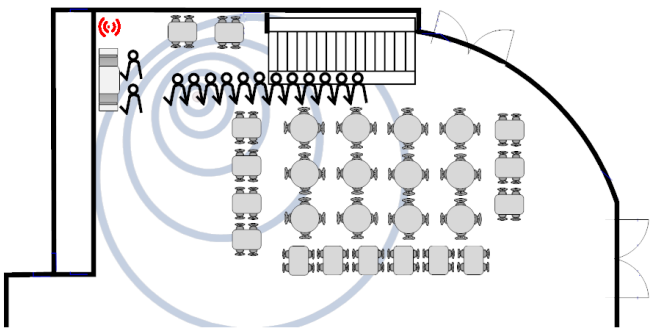


Fig. 1: Coffee shop floor plan.

1. http://ubicomp.cse.buffalo.edu/ubupdates

Although line waiting has an intrinsic problem of many venue types (such as grocery stores, banks and DMVs), every venue type has different layouts and waiting conditions. Hence, in this article we only focus on coffee shops. Particularly, we chose coffee shops due to their popularities and dynamically changing conditions; such as the variation in wait-times and number of employees.

We developed LK for a popular coffee shop at the Student Union of University at Buffalo and this article only examines the deployment and experiments of this particular coffee shop. Floor plan of the coffee shop is shown in Figure 1 [2]. The coffee shop does not have a drive-through. The customers who arrive at the coffee shop join the back of a single FIFO queue. After waiting the line, the customer is served by the staff. There are two service desks and the customer is served by either one of them. During low traffic times one of the service desk may close temporarily and only a single service desk is used. Customers who are served usually leave the coffee shop immediately. However there are some Student Union tables near the service desks and some customers sit there after picking up their coffees. There is a Wi-Fi Access Point (WAP) on the nearby wall of the line to serve customer's need for internet access. The WAP has a range of approximately 50 meters. LK utilizes BSSID of the WAP for wait-time detection.

Assumptions. LK aims to estimate the total wait-time of a customer until she is served, and does not aim to calculate neither the line length nor the service time. In reality to get a sense of how the wait-time changes over the time, we physically observed the coffee shop continuously for a week. Figure 2 shows the wait-time of the coffee shop for each 10 min. intervals for a day. As the figure indicates, wait-time fluctuates a lot during the day. Sudden increases and decreases are also prevalent. In addition, our observation show that the wait-time almost never falls below 2 minutes (i.e. min. service time) and above 20 minutes. This provides us a way to eliminate some of the false positives.

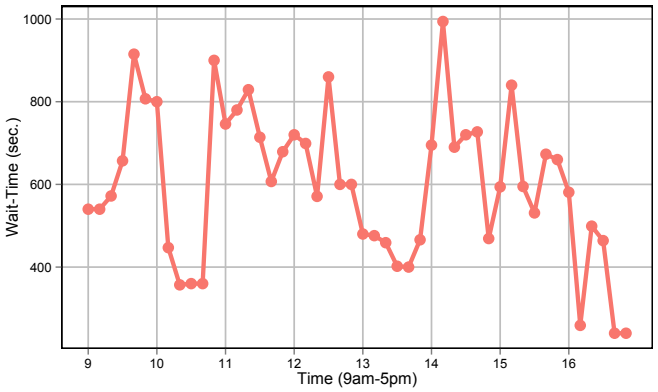


Fig. 2: Graph of actual wait-time for a day.

Finally, the wait-time detection component on the smartphones can only detect the wait-time of a customer in the coffee shop, hence, many parameters remain unknown, such

2. LK would be readily adaptable to coffee shops with similar layouts and operations

as arrival rate, service rate, service time. This prevents us from having a complete understanding of the line's operational model and introduce many challenges that need to be addressed in wait-time estimation component.

Deployment. We developed the client side of LK as a smartphone application and the server side as a service at the cloud. Although we developed both Android and iOS versions, due to the limitations imposed by iOS development framework on background tasks, in this article we only present the design, statistics and results of the Android app.

Fig. 3: A screen from the Android app

Figure 3 shows a screen from our Android app. In order to incentivize our app, we distributed fliers in the campus, post messages on Facebook and Twitter groups. As of this writing, our app has been downloaded by more than 1000 users and has been used daily by more than 300 active users. We retrieve tens of readings daily from users regarding the wait-time of the coffee shop.

3 Wait-Time Detection

The overall architecture of the system is shown in Figure 4. LK consists of two main components: the client-side component on the smartphone (which is responsible for detecting the wait-time and uploading to our server), and the server-side component in the cloud. In this section, we present the client-side component on the smartphone. The server-side component is explained in Section 5.

The client-side component includes a controller and three subcomponents: *Phone-state receiver*, *Wait-time detection unit* and *Data-Uploader*. The controller is responsible for managing and handling the interactions between these subcomponents. We explain each subcomponent in detail next.

3.1 Phone state receiver

This component serves as a notification center for the application. Android provides a notification service to let apps know about various events occurring on the device, such as Boot, Reboot, Wi-Fi connected/disconnected, Wi-Fi Signal Strength Change etc. These notifications enable apps to take action based on relevant events. We exploit this notification service in order to improve the wait-time-detection subsystem.

The Phone-State-Receiver subsystem has three different receivers which are Boot Receiver, Wi-Fi State Receiver and Power Connected Receiver. In Android, receivers work as follows: First, each receiver registers itself to listen specific events occurring on the device. Whenever the registered action happens, the operating system broadcasts a special object, i.e. an Intent, and delivers the event specific information to all registered receivers. We utilize this mechanism to monitor various relevant events for our application. For example, the Wi-Fi State Receiver gets notified when the state of the Wi-Fi connection is changed: So if the user turns the Wi-Fi off, this receiver fires at the Controller to stop the Wi-Fi Tracking Service if it is running. In addition, when a user connects to a WAP, we request the network location of the user to learn her distance from the coffee shop opportunistically.

3.2 Wait-Time detection

An intuitive way to detect wait-time of a user is to track user's location continuously and timestamp the entrance and exit of user from the coffee shop. However, excessive use of location providers (i.e. GPS or network) results in rapid battery consumption, which is undesirable for smartphone users. In order to minimize the battery consumption, we embraced the following. First, due to the cost of GPS, while tracking user's location, we solely use the network location. Second, we exploit the user's distance from the coffee shop in order to dynamically determine the location-sensing frequency. Third, once we detect that the user is around the coffee shop, we use the unique BSSID of the WAP to understand user's presence at the venue. These help conserve considerable energy; impact of 2-8% in the battery stats of the Android for a person who lives nearby the campus. We note that the principles that we discuss here are derived from our previous works. For more detailed energy-efficiency evaluation, please refer to our previous works [6], [7].

More specifically, LK's wait-time detection component works as follows. LK periodically probes user's location based on the user's distance from the coffee shop. If user's distance is greater than the threshold value (which means user is currently not in the campus), we schedule the next location probing assuming that the user is driving (30 mph). We make this conservative assumption because our experiments have found that detecting user's transportation mode (driving versus walking or biking) is in fact more costly than checking network location for every 1 minute [6].

If the user's distance to the coffee shop is less than the threshold value, we start scanning WAPs around the user every 2 minutes. Thanks to the API provided by Android, it is possible to scan the WAPs around the device without connecting to them. If the unit detects the BSSID of the coffee shop, then we detect user's entrance to the place (t_1) and reduce the scanning period to 20 seconds to detect the

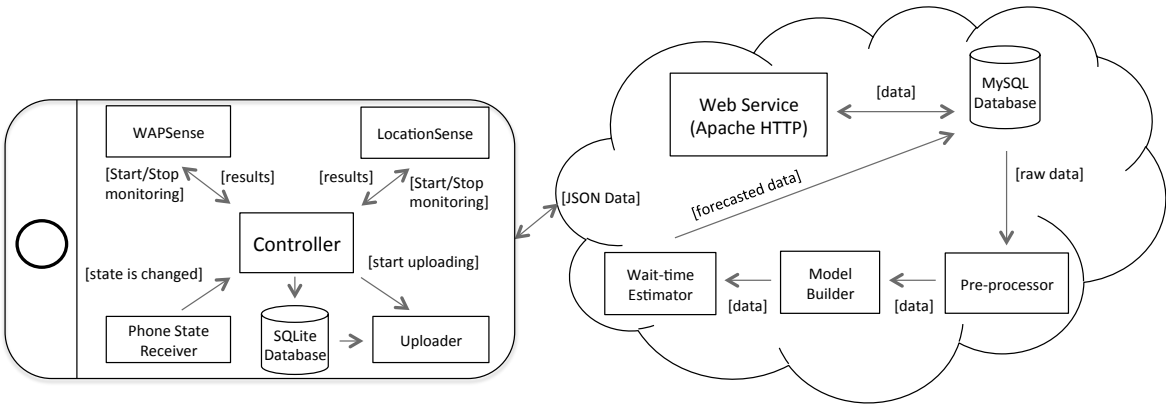


Fig. 4: Overall system architecture. Left: smartphone architecture for wait time detection. Right: cloud architecture for wait time estimation.

departure more accurately. When the BSSID is heard no longer, we assume that the user exited the coffee shop (t_2). We regard the difference of $t_2 - t_1$ as the wait-time of the user. Since we observed that most of the customers who are served usually leave the coffee shop immediately, we assume that the resulting wait-time would reflect the actual wait-time of the customers in the coffee shop (we explain a way to overcome this limitation in Section 4). In addition, while WAP scanning is enabled, we check the user's location every 5 minutes (until the user enters the coffee shop) to detect if the user left the boundary of the threshold distance (i.e. campus boundary). If so, we stop WAP scanning and start location-sensing based on the distance from the coffee shop.

3.3 Data uploader

After completing the wait-time-detection, the smartphone component tries to upload the resulting data to the cloud as an input to our wait-time estimation system. The uploading process is mostly successful in real time. However, due to the status of the device or connection, sometimes it is not possible to transmit data immediately. However, this data is still useful even if it belongs to the past. To handle this case, we have a data uploader subsystem. The data uploader is responsible for transmitting the pending wait-time detection data whenever the Phone-State-Receiver notifies the Controller about the availability of a Wi-Fi or GSM data connection.

Since a failed data transfer costs some energy, the data uploader uses some simple heuristics to increase the upload success rate. We assume that the device is charged mostly when the user is at home or office where she has a reasonably fast and reliable data connection, which is most of the time a Wi-Fi connection. Therefore, the data uploader is triggered when the device is connected to a power outlet to leverage this efficient and reliable connection. Under some circumstances, even if the device is being connected to a power outlet, it may not have such data connection available. If so, then the data uploader periodically (once an hour) checks for a data connection.

Data uploader stores the pending transfers inside a database that resides on the device. The data is sent to server as a JSON object using HTTP POST. Once the data is successfully sent, which is confirmed by a response from the server side, then the Data uploader clears up the database in order to save some storage on the device.

4 Wait-time detection using activity recognition

Our initial design assumes that most of the customers leave the coffee shop after getting their orders. Also, we remove the wait-time which is less than 2 min. and longer than 20 min. Since, users who pass nearby of the coffee shop usually spend less than 2 min. and the customers who seat at the coffee shop usually spend more than 20 min., according to our wait-time detection approach most of the time they do not constitute a problem. However, some customers may stay in between 2 to 20 min. and these insert false positives to our system. In addition, by removing long stays (more than 20 min.), we miss out the data that we can potentially use. Finally, in order for LK to scale any place we need a mechanism to differentiate customers who prefer to sit and the customers who prefer to-go. Therefore, in this section, we discuss how to extract the actual wait-time regardless of customers' willing to stay or leave. Below, we explain how we achieve this.

In order to detect the actual wait-time of a customer, we need to distinguish line waiting from other activities such as walking and sitting in the coffee shop. We utilize activity recognition [8], [9] in order to detect the actual wait-time of the user. However, we would like to emphasize that our aim is to correctly identify the wait-time of the user, not to recognize the activities that the user make while in the coffee shop. As we show in the remaining of the section, even in the presence of wrongly classified activities we were able to detect the wait-time of the user accurately.

We find that we can reliably detect the wait-time by utilizing only the accelerometer sensor on the smartphones while spending minimum battery. Figure 5a shows a representative graph for the l^2-norm (i.e. Euclidean norm: length of 3-dimensional $[x, y, z]$ vector) of accelerometer readings of a customer while in the coffee shop. As shown in the figure, during its stay in the coffee shop, a customer can be in one of the 4 different

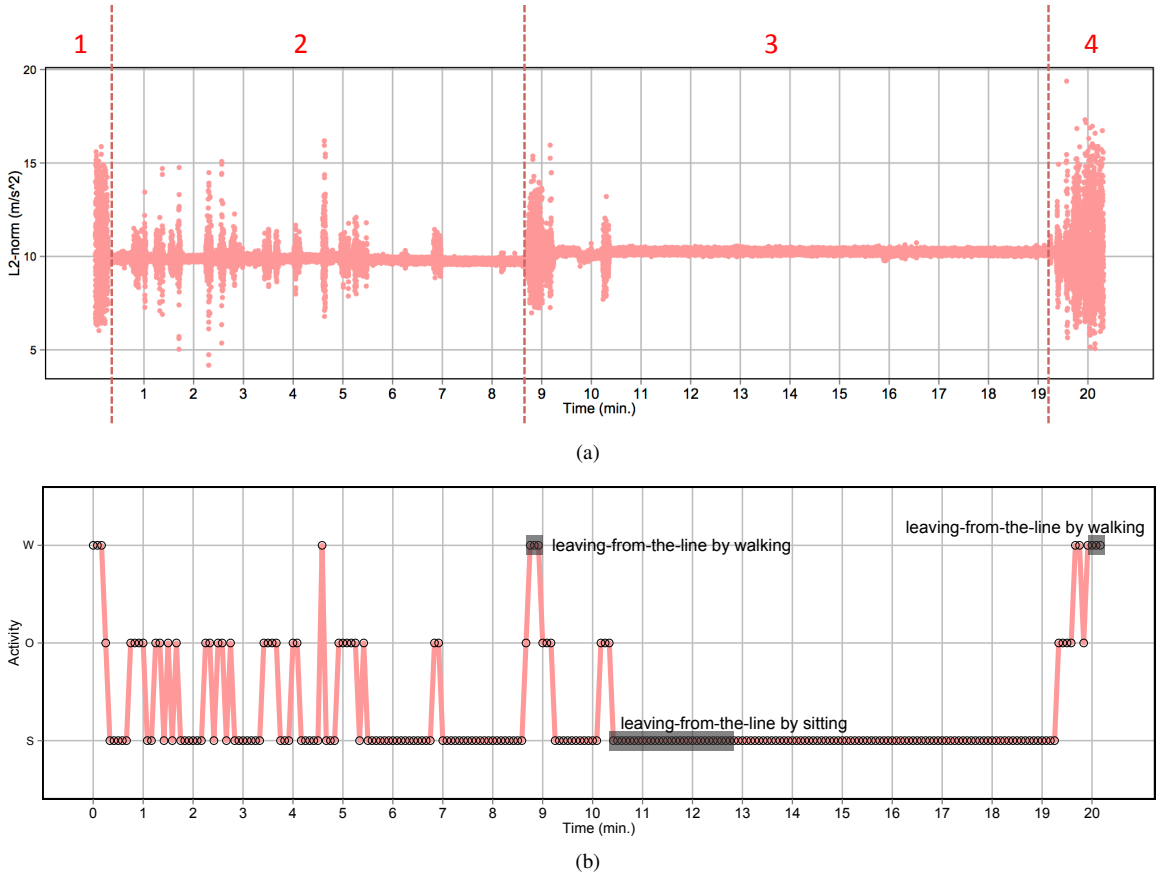


Fig. 5: a) Four regions of a graph of l^2-norm of accelerometer readings while a customer is in the coffee shop. b) Classified base events for each 5 seconds, annotated with *leaving-from-the-line* points.

regions. First is the region when the customer enters into the range of WAP. In this region customer is walking into the end of the line. High variance in accelerometer readings in this region is prevalent and indicates that the user is walking. This region may or may not be present depending on the accuracy of the presence detection component (i.e. particularly the period of WAP-sense component as explained in 3.2). In the second region, consecutive stationary (very low variance) and low-speed walking (medium variance) is observed and indicates the presence of line waiting activity. In this region, occasional walking (high variance) is allowed as the customer may walk in the line for a short period. However, in general walking in this region is short and hence results in low-speed walking activity. Then in the third region, mostly stationary (low-variance) behavior is observed. In this region, customer is actually sitting at the coffee shop. Note that while customer is sitting at the table, she may play with her phone and therefore may result in similar variance as happened in low-speed walking activity. However, most of the time user left her smartphone either in his pocket or on the table due to eating/drinking and therefore results in low-variance. Also note that third region may not present if the customer picked the order to-go (which is predominantly the case). In the final fourth region, there is a sharp increase in the variance, this indicates that the user is walking again. After a short period, user is out of the range of the WAP which indicates that the user left the coffee shop.

Based on these observation, we classified user's activity in every 5 sec. to one of the 3 base events: *Stationary (S)*, *Walking (W)* and *Others (O)*, using variance as the main feature (see Figure 5b). We then form a sequence based representation (similar to that in [10]) of the user's activities while the user is in the coffee shop. Then, our classifier look for the *leaving-from-the-line* point which is usually the point where user performs at least 15 sec. of consecutive high-variance (walking) activities. We also consider the cases where the tables are very close to the service area. In this case, customers usually sit the table in a short period (less than 15 seconds). In those cases, we look for low-variance activity (stationary) which lasts more than 2.5 minutes. Starting point of this low-variance activity is what we call *leaving-from-the-line* point again. Lastly, we assume that the customers enter to the coffee shop and directly go to the line without sitting at the beginning. We confirm this with our observation that this is the case almost all of the time and therefore a reasonable assumption to make. Moreover, after entering into the range of WAP, we assume that the customers enter to the line when they stop high-variance activity. In this way, we eliminate the time difference of entering into the range of WAP and walking into the end of line in wait-time detection. We also consider the cases where user turns on and off the screen of her smartphone. Since those are usually the moments of the

transition periods, i.e. taking from the pocket to hand or vice versa, we ignore those periods when deciding *leaving-from-the-line* point. In this way, we aim to eliminate the temporary unexpected variation caused by user's use of the smartphone while in the line. Figure 5b shows the classified base events for every 5 sec. along with the detected *leaving-from-the-line* points for Figure 5a. Below, we explain the base event detection and wait-time extraction in details.

Base Event Detection. Once the user enters into the range of WAP, we start to record the accelerometer readings. Recording continues until we detect the *leaving-from-the-line* point. We request at a sampling frequency of 50 Hz approximately. For each t seconds interval ($t = 5$ in our implementation), we have a sequence of timestamped raw accelerometer readings. In order to deal with orientation issues, we first compute the $l^2 - norm$ of the raw data. Then, we calculate the variance for every t seconds and classify the activity based on this value (We trained our model by visiting the coffee shop several times and note the different variance values). If the variance turns out to be high, then we classify the activity as W. If the variance is very low then we classify it as S. Finally, if the variance is in between the high and low thresholds, we classify the activity as O, which could be low-speed walking in the line or being stationary while playing with phone.

Extracting wait-time. As we explained earlier, we assume that the user leaves the line if she performs more than 15 sec. of consecutive walking activities or more than 2.5 min. of consecutive stationary activities (which signifies that user is no longer in the line and sitting). With the later, we try to capture the cases where users is very close to the table and just sits without walking too much. Also note that 2.5 min. is more than enough for the line we are interested in (due to high turnover), however it can be easily adjusted based on the line or can also be automatically determined by examining the accelerometer sensor readings. To extract the wait-time, we keep a sequence of streaming base events for the last 2.5 minutes and check the conformity in real-time. If we realize that user is no longer in the line, we stop accelerometer recording. Note that even in the presence of a short unexpected behavior or wrongly classified base event, since we are looking for consecutive events, our method will be able to extract wait-time correctly.

Accuracy. In order to verify the accuracy of our approach, we have collected 2 months of accelerometer data (along with the screen on/off status) from 5 Nexus S 4G devices, each one of them runs Android Jelly Bean 4.1. This data is collected by the members of our lab and we asked participants to record the entrance, exit time and the line-waiting start/end time in minutes. Later, we examine each file by visualizing it and find out the exact entrance and departure time based on the input from the participants.

During the 2 months period, we have collected 268 files in total, consisting of accelerometer readings and screen status for each visit to the coffee shop. We try to exhibit a diverse behavior during the collection process, such as using the phone during the line-waiting, sitting after the order etc. Mean *presence time* of the collected data is 11.3 minutes with 106.81 min. maximum. On the other hand, mean *actual line wait-time* of the collected data reported by the participants is approximately 6.83 min. with 18.85 min. maximum.

Using the variance-based classification algorithm described above, we calculate the actual wait-time for each visit to the coffee shop. Our experiment indicates that the mean absolute error of our wait-time detection system is 47 sec. with a median error of 20 sec. Figure 6 shows the cumulative error distribution function for our experiment. As shown in the figure, almost 85% of the errors are less than 1 min. In comparison to previous work [5], which reportedly has 10 sec. mean error, we believe that our variance-based classifier is accurate enough for our end goal of estimating the wait-time for newly arrived customers without requiring to place special equipments in the coffee shop.

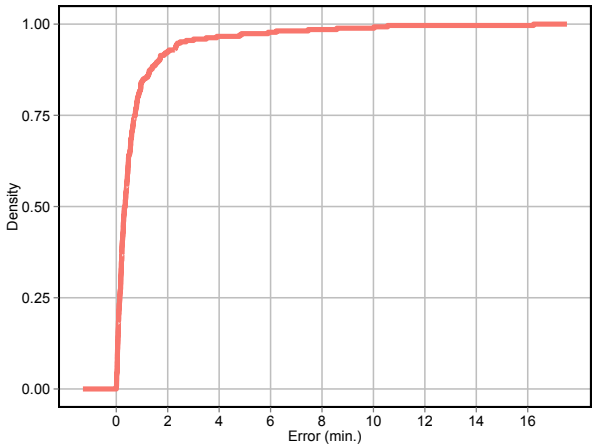


Fig. 6: CDF of errors for wait-time detection using activity recognition. 85% of the errors are less than 1 min.

5 Wait-Time Estimation

In this section, we present the wait-time estimation component of LK. This component resides on the server-side (hosted on AWS EC2 cloud for scalability) and consists of four main components: *Web service*, *Pre-processor*, *Model-builder* and *Wait-time estimator*. The web service serves as the interface between smartphones and the back-end. It accepts wait-time collected from the smartphones and provides wait-time estimation for the querying smartphones. Data collected from web service is fed into pre-processing module which is responsible mainly for removing outliers. After pre-processing, model builder builds a model periodically from all the collected data. Lastly, the wait-time estimator module uses the model and estimates the future wait-time. Below we first describe wait-time estimation problem, then we outline our methods for wait-time estimation. Finally, we present results about the performance of our methods.

5.1 Wait-time estimation problem

The problem is to estimate the line wait-time for any arriving query by using Crowdsensed data (CD), i.e. the wait-time data that is collected from the participants. Although the queries can be for anytime (past, now, future); we expect real-time querying for the current time (e.g., 5-10 minutes in to the

future) to be most useful. Hence, the wait-time estimation models need to access the most up-to-date information in CD. Wait-times usually depend on i) the time of the day, ii) weekday vs. weekend, and iii) seasonality depending on the nature of the business. For our specific coffee shop, there is less traffic in off-school days and weekends, and slightly more traffic in certain times of a day. An estimation method should capture all of these variables accurately.

The theory of time-series analysis has been usually based on a regular uniform time-series that contain enough samples [11], [12], [13]. In our case, the data is neither complete nor uniform. Therefore, a general theory of time-series is not directly applicable on CD. However, as the popularity of the application increases and by employing techniques for filling missing data, we can overcome this challenge and build robust models to estimate wait-times.

In this article, we consider an approach that is designed to handle insufficient data and that adapts/improves as more data becomes available. We call this approach Nearest Neighbor Estimation (NNE). NNE is based on constrained nearest-neighbor search in a multi-dimensional space. This approach is dynamic and works well with non-uniform and sparse CD. We believe that our method fits well to the quickly and sometimes unpredictably changing wait-time of the coffee shop.

5.2 NNE: Nearest neighbor estimation

The main idea in this method is to predict the queried value (i.e. wait-time for a particular time) using the previous history of wait-time based on their closeness in time and similarity of values. As a strawman, we first consider returning the last uploaded data as the estimated value assuming the last one would be the most closest one to the estimated interval. We call this approach *NNE-last*. When the system has enough incoming data points, the last uploaded value can in fact reflect the actual situation of the line accurately. However, this approach may fail if the data is scarce or if the data includes false positives in it. We regard NNE-last as a baseline method in our experiments.

In our second approach, we try to identify the k nearest neighbor points for the query where similarity is defined with respect to the estimation potential. The key here is to design a similarity (neighborliness) function that minimizes the estimation error for the query. In order to realize this method, we define every data point with 3 dimensions: week, day and day-interval, $[w, d, g]$. Each data is associated by a vector $[w_i, d_i, g_i]$, where w_i stands for the week of the year and is from the domain [1,52], d_i stands for day of the week and from the domain [1,7], g_i stands for interval of the day and is from the domain [1,48] (there are 48 intervals of 10 minutes between 9am and 5pm). We use euclidean distance L_{ij} to denote the similarity measure between two vectors as shown in Equation 1.

$$L_{ij} = \sqrt{(w_i - w_j)^2 + (d_i - d_j)^2 + (g_i - g_j)^2} \quad (1)$$

Once we define the similarity metric, similar to the k-nearest neighbor algorithm in machine learning, we aim to find the k nearest neighbors for the queried data point. For this purpose, we first calculate the distance of the query to each of the labeled data points (previous history of wait-times). Then we find the minimum distanced k data points and calculate the average of their wait-time as the estimated value. We call this approach *NNE-basic*.

Regression-based optimization. Although, *NNE-basic* is simple and takes the time difference between data points into account, it does not differentiate between different dimensions of the data vector, that is, the week, day and the interval are assumed to have equal weights on the distance. As an alternative, we consider multiplying each dimension with weights (α, β, γ) that are optimized based on the previous history of wait-times of the same venue.

In statistics, it is a common practice to use regression to understand the relationship between regressand and regressors. For our case, we want to quantify the relation between the wait-time (v_i) and the data vector $([w_i, d_i, g_i])$. Here, we first assume that wait-time (v) is linearly dependent to the dimensions of the data vector $([w, d, g])$. Then, we utilize the labeled data points (previous history of wait-times) and assign the weights that optimize the regression function (as shown in Equation 2) for the labeled data. Next, we define the new similarity metric L_{ij} as the weighted euclidean distance as shown in Equation 3. Then similar to our NNE-basic approach, we find the minimum distanced k data points and calculate the average of their wait-time as the estimated value. We call this approach *NNE-regression*.

$$v_i = \alpha w_i + \beta d_i + \gamma g_i + \theta \quad (2)$$

$$L_{ij} = \sqrt{|\alpha(w_i - w_j)^2| + |\beta(d_i - d_j)^2| + |\gamma(g_i - g_j)^2|} \quad (3)$$

5.3 Estimation Results

In this part, we present the estimation results of the methods we developed. However, before that, for both NNE-basic and NNE-regression approaches, we need to choose the right k value: the number of neighboring points to consider. Given the sparseness and scarcity of the data in the first few months, it is not logical to select large values. As the data grows, we can select larger values. However, selecting large values can lead to lose the impact of recent data points. To simplify our analysis, based on our experiments, we selected $k = 5$.

Data. In order to validate the estimation accuracy of LK, in this article, we conducted an academic year of experiment from September 3, 2012 to May 3, 2013. In that analysis, we focus on the time interval 9am to 5pm every weekdays. We excluded winter break and spring break from our analysis.

During that period, we collected two types of data. First is the crowdsensed data (CD) collected from the participants as explained in Section 3. Due to the nature of crowdsensing, CD is sparse and non-uniform along the time line. We have a total of 2865 data points in our CD dataset. Figure 7 shows the monthly distribution of the data points. As shown in the figure, there is less data points on December and January due to the winter break. In addition, there is surplus of data points on April due to the new advertisement campaign along with

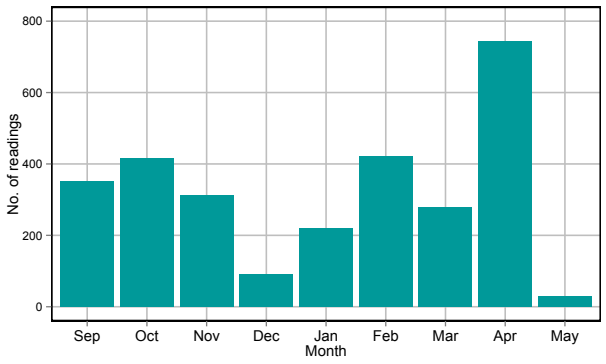


Fig. 7: Monthly counts of uploaded wait-times to our server.

Method	*MAE*	*MdAE*	*1st q.*	*3rd q.*
NNE-last	260.53	225.79	98.26	380.35
NNE-basic	210.18	180.70	87.52	290.90
NNE-regression	211.93	180.73	80.69	301.67

TABLE 1: Estimation errors (in seconds) for the experiment.

Method	*MAE*	*MdAE*	*1st q.*	*3rd q.*
NNE-last	214.86	183.23	77.94	312.75
NNE-basic	201.16	168.83	71.09	295.58
NNE-regression	192.32	158.87	73.14	279.45

TABLE 2: Estimation errors (in seconds) with more accurate wait-time detection.

the deployment of the new version of our app as explained in Section 4. We deployed the new version to collect data for testing the estimation accuracy of our activity recognition based wait-time detection approach. With the new version of our app, we were able to collect 552 extra data points. This data was collected from April 8th to May 3rd, 2013.

The second type of data we collected is the ground truth data (GD). This data is collected manually by sitting at the coffee shop and observing the customers' wait-time for every 10 minute intervals. GD is collected for the last month of our experiment (April 8th to May 3rd, 2013 for every weekday) to evaluate the estimation accuracy of LK. Due to a special event at the student union, coffee shop was closed in April 26th and therefore we excluded it from our analysis. For GD collection, every 10 minutes the experimenters (authors) pinpointed the person at the end of the line and ran a stopwatch until that person is served. As such, GD reflects the actual wait time at the coffee shop for every 10 minutes. Contrary to CD, GD is uniformly distributed along the time line. However, since we want LK to be scalable and self-bootstrapping, the estimation techniques we presented in Section 5 uses only CD. We note that we use GD only for validation purpose—to calculate estimation error of our models.

Evaluation. We evaluate LK using the ground-truth (GD) data we collected for the last month of our experiment. Since we excluded the weekends, we have 19 days in total for testing and evaluations. We evaluate the approaches using their resulting Mean Absolute Error (MAE) and Median Absolute Error (MdAE). More specifically, given a set of n ground truth wait-time: $y_1, y_2, ..., y_n$ and their estimated values (using CD): $f_1, f_2, ..., f_n$, MAE is defined in Equation 4 while MdAE is defined in Equation 5. Note that mean error is highly affected by the outliers in the data, on the other hand median error is more robust to such extremities. Therefore, we consider median error as the most representative metric for this problem. We also give the 1st (25%) and 3rd quartiles (75%) for comparison.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| \quad (4)$$

$$MdAE = median\{|f_1 - y_1|, |f_2 - y_2|, ..., |f_n - y_n|\} \quad (5)$$

Estimation results without activity recognition. Table 1 shows the estimation errors for the three different approaches using the wait-time detection method as explained in Section 3. Median error of the NNE-last is 225.79 seconds, NNE-basic is 180.70 sec. and NNE-regression is 180.73 seconds. Mean errors are approximately 30 sec. higher than the median errors. We believe that this is due to the sensitivity of MAE to the outliers which usually occur in intervals where sudden increases/decreases are observed in wait-time.

We also observed that both NNE-regression and NNE-basic are performing better than the NNE-last. We believe that sparseness of the data is the main cause of the worst performance of NNE-last. In addition, although we expected NNE-regression to overperform NNE-basic, the results show similar errors. We believe that due to the inaccuracies in detecting presence of the user (i.e. WAP scanning period), variation in service time and the false positives, we reach the point where we can improve the wait-time estimation more. Hence, NNE-regression and NNE-basic are both performing similar and results in similar errors. Next, we show that we could improve the results of NNE-regression by using a more accurate wait-time detection component.

Figure 8a and 8b show the daily MAE and MdAE of the three different approaches for the last month of our experiment (only weekdays and a day is excluded since the coffee shop was closed). Almost all of the days NNE-last is the worst performing method. On the other hand, for some particular days NNE-last perform well. We believe that this is directly related to the surplus of incoming data points in those days. On the other side, NNE-regression and NNE-basic mostly exhibit similar errors while the NNE-regression is a little better than the NNE-basic.

Figure 9a and 9b shows the comparison of ground truth and estimated values using NNE-basic and NNE-regression respectively for the day 18 and Figure 9c and 9d for the day 19. Even though we do not consider the variation in service time in our design, our estimations are close to the actual values. However, as the graphs show, sudden increases and decreases sometimes constitute a problem for both NNE-basic and NNE-regression.

Estimation results with activity recognition. In here we examine the estimation accuracy of LK using activity recognition based wait-time detection unit as explained in

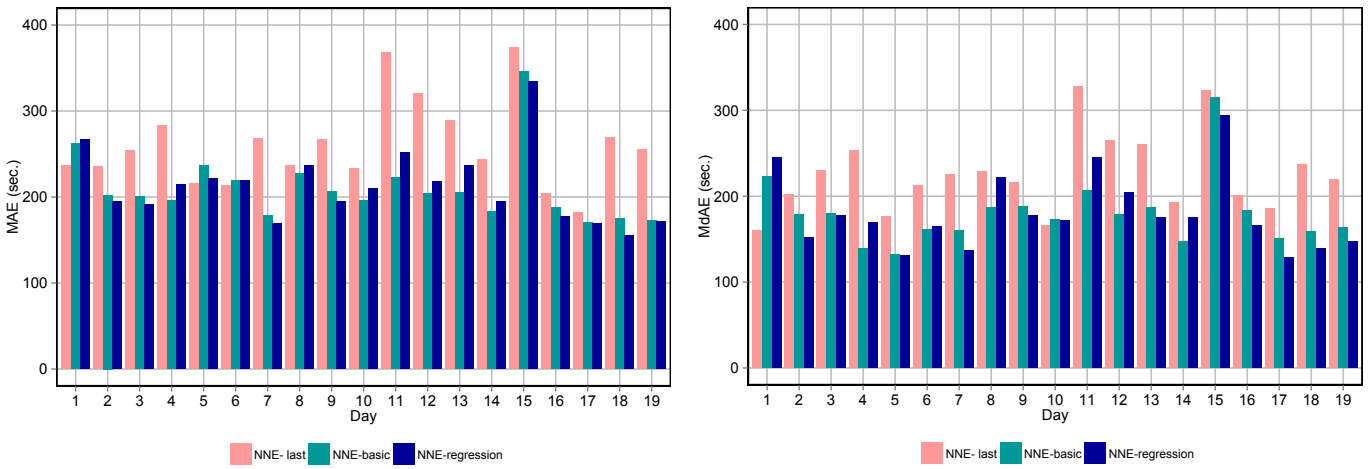


Fig. 8: a) Daily mean absolute errors. b) Daily median absolute errors.

Section 4. Since we deployed this version in the last month of our experiment, in order to have a fair amount of data, we use the first two weeks for training and the next two weeks (remaining 9 days) for testing and evaluation. Table 2 shows the estimation errors of our three different estimation approaches. Median error of the NNE-last is 183.23 seconds, NNE-basic is 168.83 sec. and NNE-regression is 158.87 sec. Similar to our previous findings, due to the outliers (mostly the sudden increases/decreases in wait-time) that are sometimes missed by our methods, mean errors are little higher than the median errors (almost 30 seconds). In overall, based on the results, NNE-regression is the best performing approach for our dataset. We believe that NNE-regression outperforms the other approaches due to its power of determining the right weights for each dimension of the data vector. In comparison to our previous finding (see Table 1), we found that new method of wait-time detection using activity recognition improved the estimation errors considerably even though we use less data (only 4 weeks) in comparison to our previous experiment. Especially the increase in the accuracy of NNE-last is prevalent. We believe that this is due to the elimination of false positives using the more accurate wait-time detection and the inclusion of more data points due to the elimination of pre-processing step on the data.

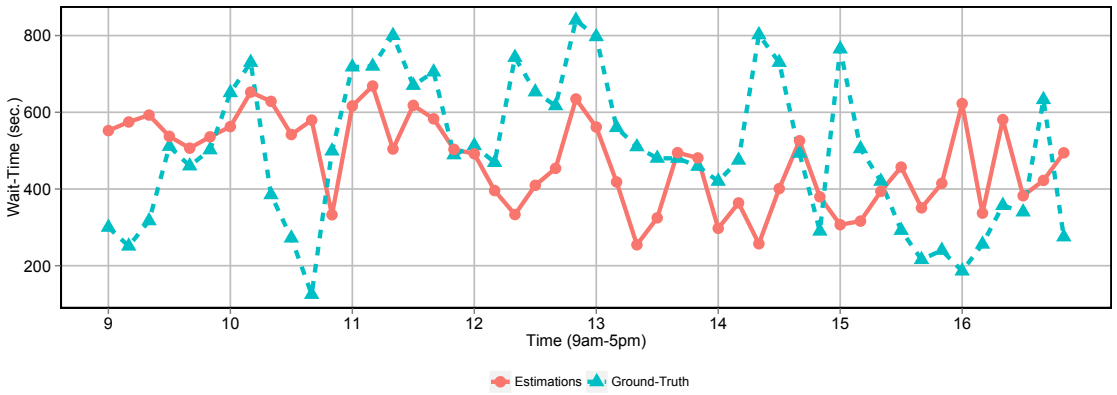
Figure 10a and 10b shows the daily MAE and MdAE of our approaches. As shown in the figure, in most of the days NNE-regression is better than the NNE-last and NNE-basic approaches. Figure 11a and 11b shows the comparison of ground truth and estimated values using NNE-basic and NNE-regression respectively for the day 12 and Figure 11c and 11d for the day 18. Similar to our previous findings, even though we do not consider the variation in service time in our design, our estimations are close to the actual values. In addition, we observed that sometimes NNE-regression fails to adopt itself to the sudden increases/decreases in wait-time while the NNE-basic and NNE-last are more successful in adapting to sudden increases and decreases. In future, we may consider to merge NNE-regression with NNE-last so that LK can adapt itself to changing conditions more accurately. However, in overall, our estimations are close to the actual values.

We believe that variation in service time for different users (which we did not consider), and the inaccuracy in detecting the user's entrance/exit to the coffee shop due to WAP scanning period are the limiting factors for better estimation results. Moreover, variation in the number of employees working in the coffee shop during the day, different speed-of-service for different employees are also other limiting factors for better estimation. However, in overall, we believe that LK performs well enough for a coffee shop whose wait-time ranges from 2 to 20 minutes by estimating the wait-time with only 2.5 min. median error.

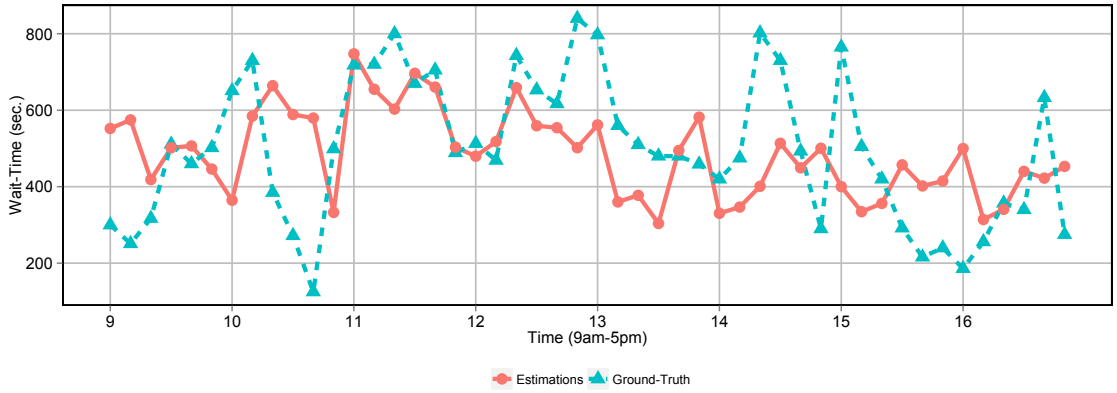
6 Scaling LK to Multiple Venues

While we presented LK's deployment for one coffee shop, we believe LK's deployment can be extended for other coffee-shops too. To add a new business to the LK, we only require the geographical locations, i.e. latitude and longitude, and the BSSID of the business. After a business is added, LK immediately starts receiving wait-time data from the users visiting that business. Depending on the number of LK users visiting the business, it may take time for LK to construct a model and start providing accurate wait-time estimations for the business. To speed up this process, a business added to LK may manually provide wait time for a week, or offer promotions and coupons for users who install the LK app and check-in frequently. As an alternative, we could use data from similar businesses to help bootstrapping effort for the newly added businesses. Below, we include more details on how to scale LK to a large set of locations.

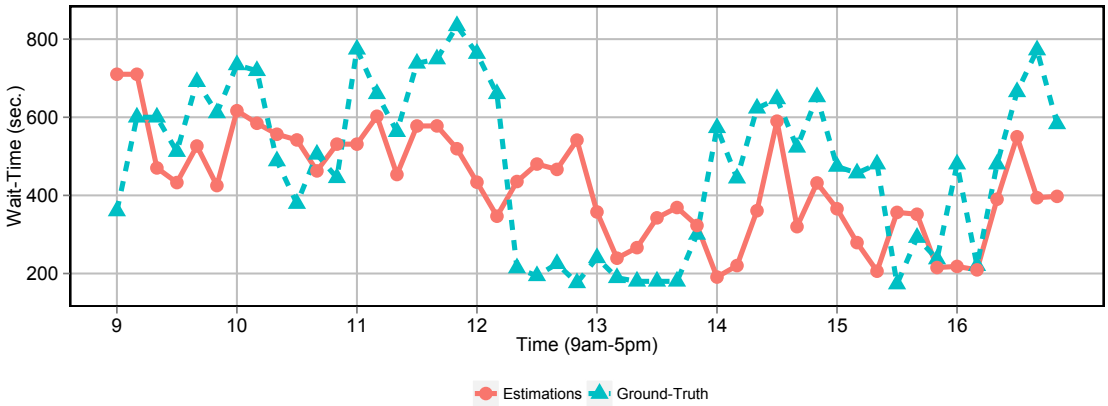
Automated learning of BSSID. In our reported deployment we manually learned the BSSID of the WAP in the coffee shop. However, in order to scale LK to other locations quickly, we can automate this process as follows. Initially when the BSSID of the WAP in a business is still unknown, LK relies on just the Location sensing mechanism for wait-time detection (this time GPS is utilized for accurate presence detection). During



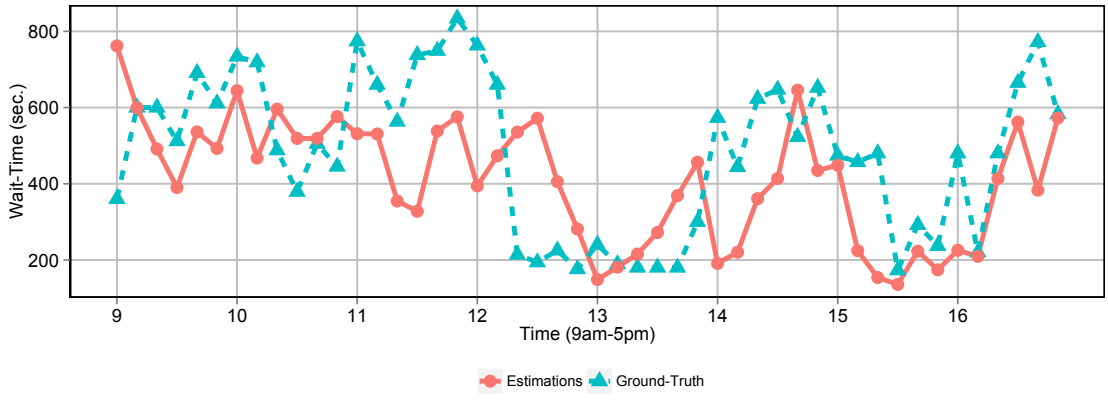
(a)



(b)

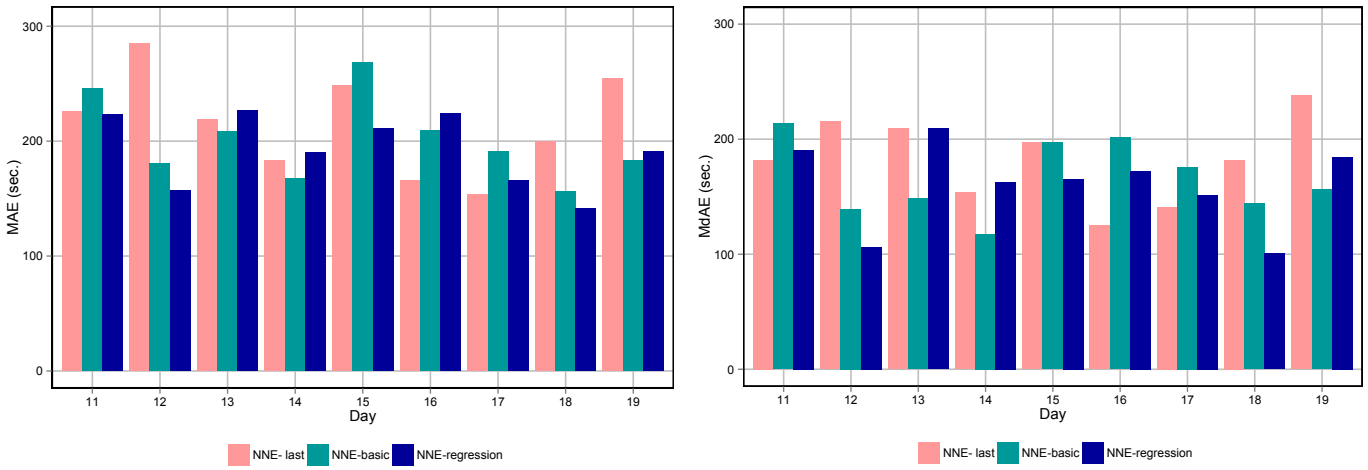


(c)



(d)

Fig. 9: Plot of ground truth vs. forecasted data using a) NNE-basic for the day no. 18. b) NNE-regression for the day no. 18. c) NNE-basic for the day no. 19. d) NNE-regression for the day no. 19.



(a) (b)

Fig. 10: a) Daily mean absolute errors. b) Daily median absolute errors.

this phase, LK app instances scan for the available WAPs in that business location and upload these to the LK servers. Learning and validating the BSSID of a business involves recurring observations of the same BSSID by different users at different times. After the BSSID of the business is learned, LK starts accepting line wait-time detections from that business via WAP as well. This increases the data collected from that business and shortens the period for constructing an accurate wait-estimation model.

Integrating LK with social networks. We plan to use social network services and APIs to quickly scale LK for line wait-time monitoring of businesses nationwide and worldwide. For example, we will obtain the geographical locations of new businesses to add to LK by using the Foursquare [14] Venue API (which does not even require a login to Foursquare). We also plan to integrate/embed LK as an extension to the existing popular location-based services such as Facebook, Foursquare and Google+.

Offline wait-time estimation. Although LK's wait-time estimation component resides on EC2 for scalability, sometimes it is desirable for users to have the estimation models on their phone for immediate and offline access to the future wait-time. Although it is not scalable to hold models for all of the franchises, it is possible for user's to have their favorite shops's model on their smartphones which is assumably a few.

7 Discussion

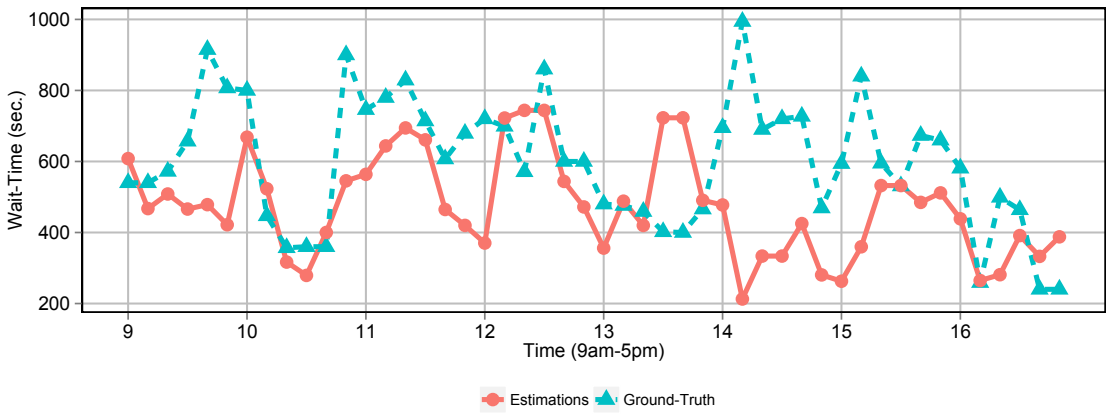
During our deployment, we faced several challenges and learned some lessons.

Device fragmentation in Android ecosystem. Android ecosystem consists of a large number of different vendors and manufacturers. Therefore, unexpectedly, same API in different phones may give different values. For example, in our line wait-time detection component, we set the accelerometer's sampling frequency to the same scale. However, we quickly noticed that this same scale gives different results even in Google's own phones; Nexus S and Galaxy Nexus. Hence, an approach to generate uniform accelerometer readings for different devices was necessary. We resolve this problem by taking time as a threshold rather than the number of sampling points accumulated. Supporting for different screen sizes was also another challenge that we encountered during our design process.

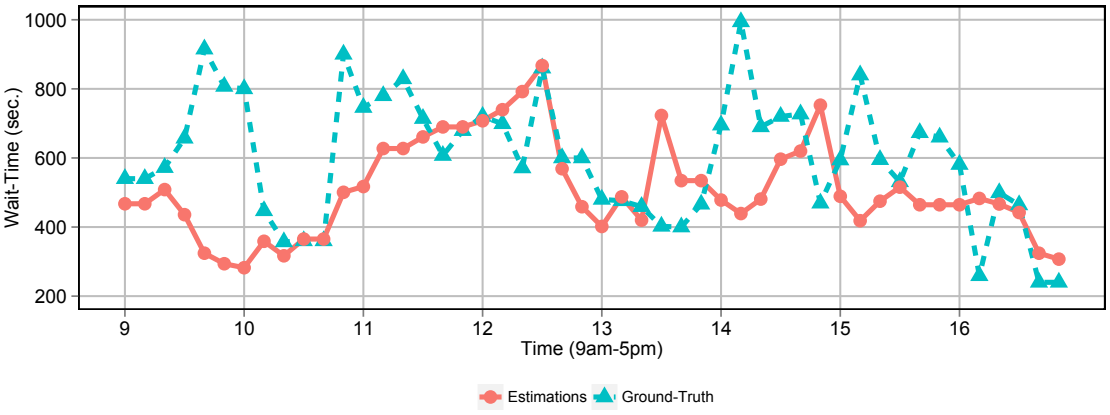
Bootstrapping. Finding enough user-base is a classical problem of any crowdsensing system. Although, LK does not require everyone in the line to have our application, we need a reasonable amount of users so that LK can give accurate line wait-time estimations. To achieve this objective, we distributed fliers and post status updates on social media such as in Facebook and Twitter for several weeks. Although, we did not incorporate any monetary incentives in our current deployment, we think that in future we may give coupons randomly while the user is in the coffee shop as an incentive for others to install our application. For similar future apps, we will also consider using the PhoneLab [15] testbed to help with bootstrapping efforts.

Design decisions. For simplicity's sake, we decided early on to handle the general case accurately, and ignore most rare corner cases which can increase the battery consumption. For example, LK's presence detection component does not use GPS and uses 2-minutes Wi-Fi scanning periods to detect user's presence at the coffee shop. We believe that this is a good tradeoff to take: inaccuracy incurred by not being so precise here provides LK to be energy-efficient and attract more users to install our application. As another example, in our activity recognition based wait-time detection component we assume that customers first go into the line before sitting at the tables. Although, there might be cases where people first sit and then go in to the line, we observe that these are very few and handling them may increase the complexity of our line wait-time detection component.

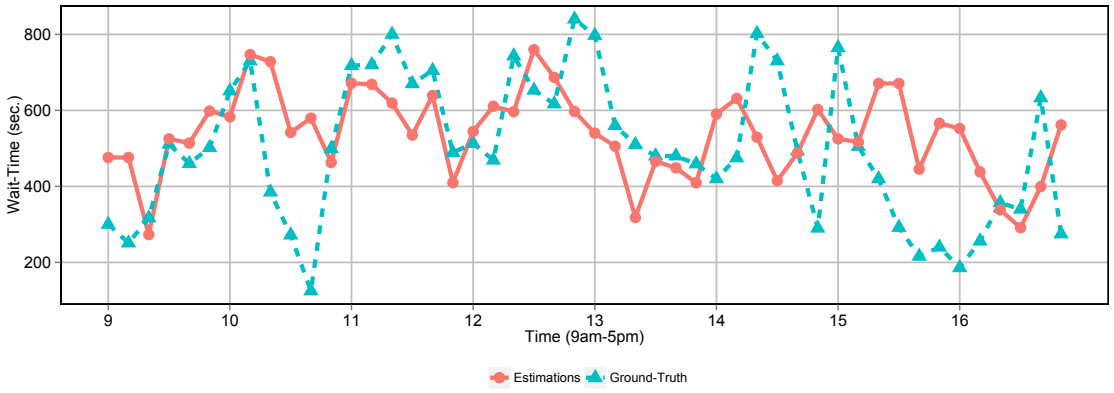
Dynamic adaptation of estimation models. The question of *"Can LK know/learn about its accuracy and later adopt its estimation methods/parameters accordingly?"* is an important one. We believe that the answer for this question is yes. To calculate the accuracy, LK can observe the difference between its estimate and the wait-time that has just been collected by



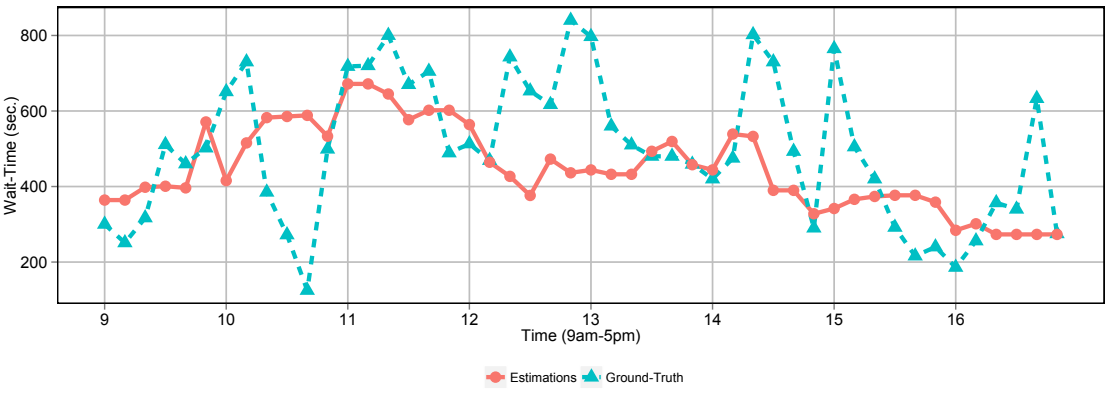
(a)



(b)



(c)



(d)

Fig. 11: Plot of ground truth vs. forecasted data using a) NNE-basic for the day no. 12. b) NNE-regression for the day no. 12. c) NNE-basic for the day no. 18. d) NNE-regression for the day no. 18.

the wait-time detection unit of LK. Based on the error, LK can switch to whichever NNE works best, or whichever k value works best. This switching can be done periodically in every day/week. We consider to incorporate this feature in our future design.

8 Related Work

Mobile crowdsensing. Participatory [16] and opportunistic [17] sensing are the two ends of the sensing spectrum where both have pros and cons and choice of which one to use depends on several factors. The former depends on users' explicit participation, while the latter does not and therefore can scale more easily. In parallel to the widespread use of smartphones in recent years, we have witnessed an increasing interest in opportunistic sensing with smartphones–which is usually dubbed as mobile crowdsensing [18]. To the best of our knowledge, LK is the first mobile crowdsensing system that explore line wait-time estimation in a coffee shop. However, there are other researchers who explore different domains of mobile crowdsensing. For example, in TagSense [19], authors leverage camera, compass, accelerometer and GPS sensors of the phones to provide an image tagging system. In [20], Bao and Choudhury introduce MoVi that employs smartphones to enable collaborative sensing using videos for recognizing socially interesting events. In [21], authors develop a smartphone-based solution to monitor road and traffic condition of a city.

Line wait-time detection and estimation. Traditional solutions to the line wait-time monitoring problem are based on infrastructure-based solutions such as camera placement [1], [2], sensor deployment [3] or monitoring signals generated through bluetooth [4] or Wi-Fi capable devices [5]. However, these solutions usually do not scale well as they are costly to deploy and mostly designed for specific places. Also most of the time, extensive training and testing are required before real-world deployment. Among these, we identified [5] as the most relevant work to ours. In [5], authors monitor the received signal strength of the smartphones using a specially placed WAP. Using the signal strength pattern resulted from user's movement, they try to identify the wait-time of the user. On the other hand, LK provides a more holistic approach, from detecting user's presence at the coffee shop to wait-time detection and wait-time estimation. In addition, LK provides an infrastructure-free solution (assuming most of the coffee shops have already WAPs) for wait-time detection by solely relying on crowd's automatically generated input. Finally in [5], the aim is to detect the wait-time of a customer who leaves the line, while in ours, we try to estimate the wait-time for a newly arriving customer.

Besides, line wait-time detection is only one part of the problem. Due to the sparse and non-uniform nature of the collected data, we need to estimate the current and future wait-times. In literature, line wait-time estimation has been explored mostly in the context of Queue Theory [22]. Those works usually assume that examiners have the full knowledge of the parameters, i.e. queue discipline, arrival rate, service rate etc. However, in our problem we only have wait-times and the associated timestamps. So queueing theory is not easily applicable for our problem. On the other hand, line wait-time estimation is related to some problems in general time series theory where the task is to forecast future data using the previous ones. Number of different techniques have been proposed in the literature ranging from ad hoc methods (i.e. moving average, exponential smoothing) to complex model-based approaches which take trend and seasonality into accounts (i.e. Decomposition, Holt-Winters, ARIMA) [11], [13], [12]. A major challenge is that general time series analysis depends on data that is uniformly distributed along the time. However, our application has non-uniform and initially sparse data that prevent us from applying the general time series theory easily.

Localization. LK needs to achieve energy-efficient localization in order to detect user's presence at the coffee shop. In the literature, both localization and power-aware sensing are explored. In [23], authors examine the human localization in a building using smartphone sensors and randomly placed audio beacons in the building. In UnLock [24], authors propose an unsupervised indoor localization by exploiting an identifiable signature in one or more sensing dimensions such as the pattern of accelerometer readings while in an elevator. Similarly, in LK we use the unique BSSID of WAP in the coffee shop in the interest of localization. Finally, in [25], authors identify four factors that waste energy: static use of location sensing mechanism, absence of use of other sensors, lack of cooperation among applications, and finally ignoring battery level while sensing. LK uses dynamic location sensing based on the user's distance from the coffee shop to achieve energy-efficiency.

Activity Recognition. Due to the proliferation of sensors in commodity mobile devices, identifying the physical activity of a user has recently gained attention in pervasive community. Aside from using sensor motes to recognize user's activity [26], there has been an increasing interest on using smartphones to perform activity recognition. In [8] authors use accelerometer in smartphones to recognize different activities including walking, jogging and standing. In [9], authors use smartphones to determine transportation mode of a user. In most of these works, researchers utilize accelerometer along with some other sensors such as GPS. As explained in Section 4, LK exploits accelerometer on smartphones for more accurate wait-time detection.

9 Conclusion and Future Work

In this article, we presented LineKing, a novel crowdsensing line wait-time monitoring system. LK alleviates the limitations of the infrastructure-based solutions by proposing an automatic and accurate wait-time detection on the smartphones along with accurate wait-time estimations in the cloud. We deployed LK in a coffee shop of the University at Buffalo, SUNY. LK has been available for more than a year and has been downloaded by more than 1000 users. Our experiments indicated that we can estimate the actual wait-time of the coffee shop with 2-3 minutes median absolute error.

For future work, we consider several extensions of LK. In our current design, we focused on the total wait-time

of a user without considering the service time. However, detecting the service time could be useful for both the venue owners and the customers. As a second extension, LK can be extended to report the line length along with the current wait-time. As shown in the Figure 5a, when the customer is waiting in the line, she generates peaks which mostly occur when another person has left the line. Therefore counting such peaks in the accelerometer graph roughly corresponds to the number of customers in the line. As another extension, we consider scaling LK's estimation component to support multiple (possibly hundreds) venues at the same time. For this, LK needs to handle the incoming stream of big data while forecasting the wait-times of different venues accurately.

REFERENCES

[1] MultiQ at Shanghai World Expo 2010,http://www.multiq.com/wp-content/uploads/2013/05/case_study_shanghai_world_expo_2010.pdf.

[2] IRISYS Queue Management, http://www.irisys.co.uk/queue-management/.

[3] D. Bauer, M. Ray, and S. Seer, "Simple sensors used for measuring service times and counting pedestrians," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2214, no. -1, pp. 77–84, 12 2011. [Online]. Available: http://dx.doi.org/10.3141/2214-10

[4] D. Bullock, R. Haseman, J. Wasson, and R. Spitler, "Automated measurement of wait times at airport security," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2177, no. -1, pp. 60–68, 12 2010. [Online]. Available: http://dx.doi.org/10.3141/2177-08

[5] Y. Wang, J. Yang, Y. Chen, H. Liu, M. Gruteser, and R. P. Martin, "Tracking human queues using single-point signal monitoring," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14. New York, NY, USA: ACM, 2014, pp. 42–54. [Online]. Available: http://doi.acm.org/10.1145/2594368.2594382

[6] M. F. Bulut and M. Demirbas, "A holistic approach for energy efficient proximity alert on android," in *2013 IEEE Global Communications Conference, GLOBECOM 2013, Atlanta, GA, USA, December 9-13, 2013*, 2013, pp. 2816–2821.

[7] M. F. Bulut, Y. S. Yilmaz, M. Demirbas, N. Ferhatosmanoglu, and H. Ferhatosmanoglu, "Lineking: Crowdsourced line wait-time estimation using smartphones," in *MobiCASE*, 2012, pp. 205–224.

[8] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, no. 2, pp. 74–82, Mar. 2011.

[9] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 13:1–13:27, Mar. 2010.

[10] R. Hamid, S. Maddi, A. Johnson, A. Bobick, I. Essa, and C. Isbell, "A novel sequence representation for unsupervised analysis of human activities," *Artif. Intell.*, vol. 173, no. 14, pp. 1221–1244, Sep. 2009.

[11] C. Faloutsos, "Mining time series data," in *SBBD*, 2005, pp. 4–5.

[12] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ser. ICDM '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 273–280.

[13] O. Maimon and L. Rokach, Eds., *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Springer, 2010.

[14] Foursquare venues platform, https://developer.foursquare.com/overview/venues.

[15] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen, "Phonelab: A large programmable smartphone testbed," in *Proceedings of First International Workshop on Sensing and Big Data Mining*, ser. SENSEMINE'13. New York, NY, USA: ACM, 2013, pp. 4:1–4:6.

[16] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *In: Workshop on World-Sensor-Web (WSW06): Mobile Device Centric Sensor Networks and Applications*, 2006, pp. 117–134.

[17] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson, "People-centric urban sensing," in *Proceedings of the 2nd annual international workshop on Wireless internet*, ser. WICON '06. New York, NY, USA: ACM, 2006.

[18] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[19] C. Qin, X. Bao, R. R. Choudhury, and S. Nelakuditi, "Tagsense: Leveraging smartphones for automatic image tagging," *IEEE Transactions on Mobile Computing*, vol. 13, no. 1, pp. 61–74, 2014.

[20] X. Bao and R. R. Choudhury, "Movi: mobile phone based video highlights via collaborative sensing," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 357–370.

[21] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 323–336. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460444

[22] R. B. Cooper, *Introduction to Queueing Theory*, 2nd ed. New York, NY: North-Holland, 1981.

[23] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury, "Did you see bob?: human localization using mobile phones," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, ser. MobiCom '10. New York, NY, USA: ACM, 2010, pp. 149–160.

[24] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 197–210. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307655

[25] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10, 2010, pp. 315–330.

[26] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection," in *Proceedings of the 5th European conference on Wireless sensor networks*, ser. EWSN'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 17–33.

Muhammed Fatih Bulut received his B.S. degree from Computer Engineering Department of Bilkent University, Turkey in 2009 and Ph.D. degree from University at Buffalo, SUNY in 2014. His research interests include mobile computing, crowdsourcing and social networks. More specifically his research focuses on employing smartphones and social networks to build innovative people-centric, socially-aware systems and applications.

Murat Demirbas received his B.S. degree in Computer Science and Engineering from Middle East Technical University (METU), Ankara, Turkey. He received his MS in 2000 and PhD in 2004 from The Ohio State University. His research interests are in the broad area of distributed and networked systems, and span the areas of cloud computing, distributed algorithms, fault-tolerant computing, self-stabilization, ubiquitous computing, wireless sensor networks, smartphones, crowdsourcing.

Hakan Ferhatosmanoglu received his B.S. degree from Bilkent University, Ankara, Turkey in 1997 and Ph.D. degree in Computer Science from University of California, Santa Barbara in 2001. His research interests include scalable management and mining of multi-dimensional data. He received Career awards from the US Department of Energy, US National Science Foundation, and Turkish Academy of Sciences.