LineKing: Crowdsourced Line Wait-Time Estimation using Smartphones

Muhammed Fatih Bulut[1], Yavuz Selim Yilmaz[1], Murat Demirbas[1], Nilgun Ferhatosmanoglu[2], Hakan Ferhatosmanoglu[3]

[1] University at Buffalo, SUNY, Buffalo, NY 14260, USA
`{mbulut,yavuzsel,demirbas}@buffalo.edu`
[2] THK University, Ankara 06790, Turkey
`nilgunf@thk.edu.tr`
[3] Bilkent University, Ankara 06800, Turkey
`hakan@cs.bilkent.edu.tr`

Abstract. This paper describes the design, implementation and deployment of LineKing (LK), a crowdsourced line wait-time monitoring service. LK consists of a smartphone component (that provides automatic, energy-efficient, and accurate wait-time detection), and a cloud backend (that uses the collected data to provide accurate wait-time estimation). LK is used on a daily basis by hundreds of users to monitor the wait-times of a coffee shop in our university campus. The novel wait-time estimation algorithms deployed at the cloud backend provide mean absolute errors of less than 2-3 minutes.

Key words: Crowdsourced sensing, Smartphone applications, Wait-time estimation

1 Introduction

Long and unpredictable line lengths at coffee shops, grocery stores, DMVs, and banks are inconveniences of city life. A webservice that provides real-time estimation of line wait-times would help us make informed choices and improve the quality of our lives. While a line wait-time estimation service may first be regarded as a toy or luxury, it is worth recalling that webservices that we now categorize as necessities (e.g., maps, online shopping, social networks, and mobile internet) have also been perceived as similar initially. Moreover, understanding waiting line has benefits beyond improving the end-user experience because this has been a long standing problem in the operations research area.

Our method to address the line wait-time detection problem is crowdsensing with smartphones. In the very first prototype of our service, we asked users to manually provide line wait-times when they are waiting in line and tried to serve other users with the data input by these. We quickly noticed that this is an extra work for the users and the data arriving from the volunteers is too scarce to serve good results to the queriers. In the later versions of our service, we automated the line wait-time detection by using the localization capabilities of the smartphones in an energy-efficient manner, which we detail in this paper.

Line wait-time detection is, however, only one part of the problem. We found that even when our automated line 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-times. To address this problem we applied statistical techniques (heuristic regression, exponential smoothing and Holt-Winters method) to the line wait-time data we collect. This allowed us to learn patterns from the current and historical data to provide accurate responses to queries.

Our contributions are as follows:

1. We designed, implemented, and deployed a crowdsourced line wait-time estimation system called LineKing (LK). Our LK apps [1] for Android and iPhone platforms are downloaded by more than 1000 users in our university, and are used on a daily basis by hundreds of users to monitor the line wait-times of a coffee shop in the student union of our university. To the best of our knowledge, LK is the first crowdsourced line wait-time estimation service.
2. As part of LK, we implemented a fully automatic, energy-efficient, and accurate wait-time detection component on Android and iOS platforms. This component uses new domain specific optimizations to achieve better accuracy and performance.
3. For the wait-time estimation problem, we introduced a novel solution based on a constrained nearest-neighbor search in a multi-dimensional space. Then, we improved it by adapting two statistical time-series forecasting methods, namely exponential smoothing and Holt Winters, and demonstrated the strengths and weaknesses of these solutions.
4. We collected and analyzed several months of line wait-time data, which can be basis for other work on similar topics. To extend on our current work, we also discuss the challenges and opportunities for scaling LK to online monitoring of line wait-times over many venues across the world.

Outline of the rest of the paper. We discuss related work next and describe the model and limitations of our deployment in Section 3. We present the line wait-time detection component of LK in Section 4 and the line wait-time estimation component in Section 5. Section 6 presents the results from our deployment and experiments. In section 7, we discuss techniques to scale LK to other coffee shops and franchises.

2 Related Work

2.1 Smartphone sensing

The increasing computational power and sensor capabilities of the smartphones resulted in increasing interest on smartphone sensing [16, 9]. In TagSense [21], authors leverage camera, compass, accelerometer and GPS sensors of the phones

[1] `http://ubicomp.cse.buffalo.edu/ubupdates`

to provide an image tagging system. In [2], Bao and Choudhury introduce MoVi that employs smartphones to enable collaborative sensing using videos for recognizing socially interesting events. In [20] Miluzzo et al. utilize smartphone sensors to infer user's status and share on Facebook.

A significant portion of smartphone sensing focuses on location-sensing where both localization and power-aware sensing are explored. In [4], authors examine the human localization in a building using smartphone sensors and randomly placed audio beacons in the building. In [22], 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. The paper analyzes aforementioned factors and proposes a middleware location sensing framework which adaptively a) toggles between GPS and network based on the accuracy of the providers, b) suppresses the use of location updates based on the context (i.e. user is stationary or moving), c) piggybacks on other applications' location sensing mechanism and d) changes parameters of location updates based on the battery-level. However, that work does not focus on region monitoring and does not take distance into account as a factor for selecting the mode for providing localization. In LK we use distance from the point of interest for toggling the mode for providing the localization.

2.2 Line wait-time estimation

Line wait-time estimation has been explored mostly in the context of Queue Theory [5, 17]. Those works assume that examiners have 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.

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) [7, 19, 12]. A major challenge is that general time series analysis depend on data that is uniformly distributed along time. However, our application has non-uniform and initially sparse data that introduce new challenges.

3 Model and Assumptions

We deployed LK at a popular coffee shop at the Student Union of University at Buffalo. Floor plan of the coffee shop is shown in Figure 1. 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. Our detection system utilizes BSSID of the WAP for wait-time detection.

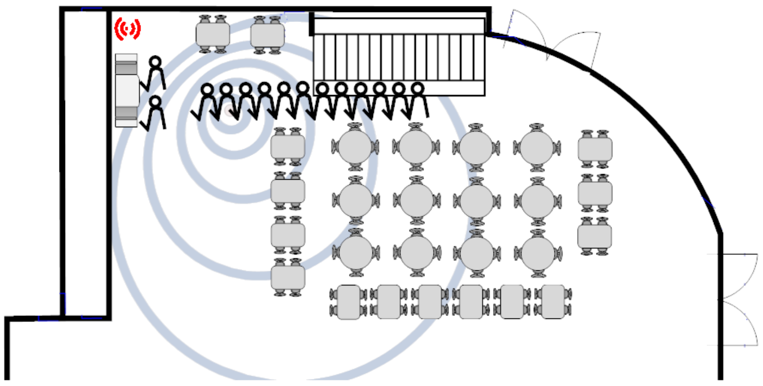
Fig. 1. Coffee Shop Floor Plan

LK aims to estimate the total wait-time of a customer, and does not aim to calculate neither the line length nor the service time. Moreover, our wait-time detection component on the smartphones cannot differentiate between the seated customers and the customers who wait in the line. Therefore, there are two sources of false-positives in our system: 1) a customer who seats after being served, and 2) a customer who takes a look at the coffee shop without waiting in the line. To get a sense of how the wait-time changes over the time, we physically observed the coffee shop continuously for one week. Our observations show that the wait-times almost never fall below 2 minutes (i.e. min. service time) and above 20 minutes. We use this information to eliminate false-positives. Although some false-positives are eliminated this way, customers who sit between 2-to-20 minutes still insert false-positives. Based on our observation sitting customers are the minority with respect to all customers, and our data-analysis techniques manage to filter their data as noise (see Section 5). Later, in Section 7, we explain a way of differentiating seated customers from others to further increase the accuracy of our detection component.

The wait-time detection component on the smartphones can only detect the total wait-time a customer spent in the coffee shop, hence, many parameters remain unknown, such 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.

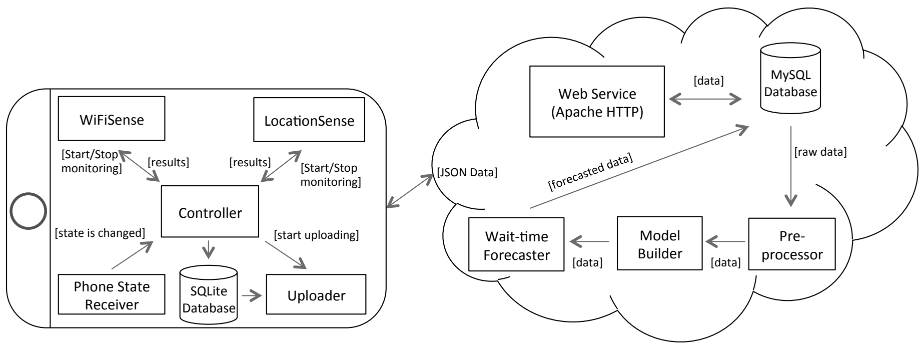

Fig. 2. Overall system architecture. Left: smartphone architecture for wait time detection. Right: cloud architecture for wait time estimation.

4 Smartphone Design

The overall architecture of the system is shown in Figure 2. LK consists of two main components: the client-side component on the smartphone, and the server-side component on the cloud. In this section, we present the client-side component on the smartphone. The server-side component is explained in the next section.

The client-side component includes a controller and three subcomponents: Phone-State-Receiver, Wait-Time-Detection (LocationSense+WiFiSense), and Data-Uploader. The controller is responsible for managing and handling the interactions between these subcomponents. We explain each subcomponent in detail next.

Remark: In the following sections, we describe the smartphone component designed for the Android Operating System [1]. Due to the development limitations that iOS imposes, some of the features below are not available for iOS [11]. We will refer to such features in the text to distinguish those parts.

4.1 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, Proximity Alert, Wi-Fi Signal Strength Change etc. iOS also has similar functionalities such as Significant Location Change action. 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 Proximity Alert 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. Another example is the Proximity Alert Receiver which notifies our app of entering and exiting the coffee shop. We explain this alert in detail next.

How does proximity alert work?
Proximity alert is a service provided by the Android OS (Region Monitoring for iOS is also present) that periodically checks the location of the device and fires alerts for the entering and exiting events for a specified geo-fenced region. The programmer can set the proximity alert by providing a location (i.e., latitude and longitude) and a radius, which represents a circle around the location. On the other hand, the device also has a location and its location has an accuracy. Hence, while one circle denotes the geo-fence that the application tracks, another circle denotes the device's location. When these two circles intersect *entered* event is fired, and when they are separated *exited* event is fired.

A natural way to detect wait-time is to set proximity alert for the coffee shop and use entered and exited events to get notified. However, continuous use of proximity alert drains the battery quickly. Figure 3 shows the battery consumption of an Android device which registers for only one location compared to another Android device with no proximity alert. Even if we don't move the device (same location), the former drained the battery in 18 hours compared to almost 75% remaining battery level on the latter device.

Due to the costs of proximity alert (especially in Android), we do not use proximity alert directly. As we explain in the next section, our system sets a proximity alert only if the user is close to the targeted location. Otherwise, our system adaptively checks if the user is making progress towards the coffee shop by monitoring the distance.

4.2 Wait-time detection

The wait-time detection subsystem uses the device location and the BSSID of the Wi-Fi Access Point (WAP) to detect the user's presence at the coffee shop. Due to the high energy cost of continuous sensing, we adopt a hint-based approach to initiate sensing. We utilize the following two hints: First, we assume that if a user opens the app to check the line wait-time, then she is a potential candidate to visit the coffee shop soon. Second, we utilize the user's coarse-grained location

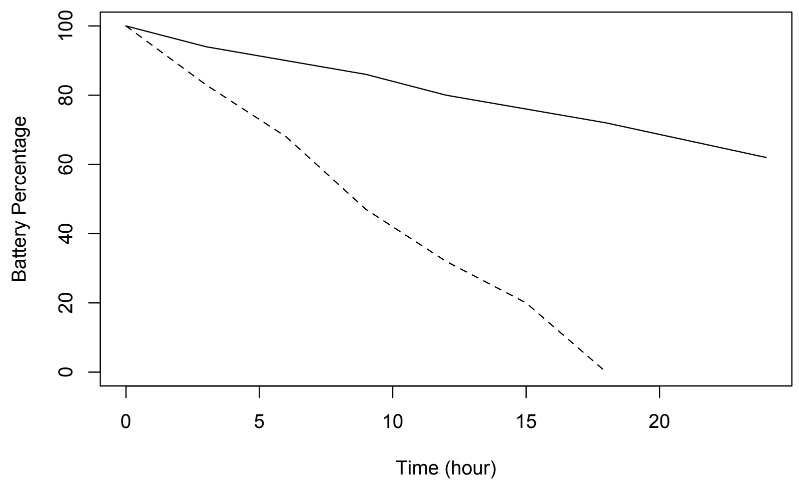

Fig. 3. The effect of using proximity alert continuously. The dashed line represents the battery level of a device which registers for one proximity alert, and the solid line represents a device which registers for none.

and start/stop monitoring if the user is close to the coffee shops. By using these two hints LK achieves energy-efficient monitoring.

LK employs two alternative methods to detect the wait-time at a coffee shop: Location-sensing and WiFi-sensing. Both methods are orthogonal and return results with similar accuracies, which makes the two methods replaceable. If the coffee shop does not have a WAP (or the WAP is not learned/validated yet), then the Location-sensing can be used. If the coffee shop is inside a big mall where the Location-sensing does not work accurately, then the WiFi-sensing is more advantageous. In our deployment we used both methods, and we provide results from both in the experiments section. We describe these two techniques next.

Location-sensing Once the smartphone component has a hint that the user may go to a coffee shop, it dispatches a new job and starts monitoring the user's location. First the distance between the coffee shop and user's current location is calculated. If the user is close enough (i.e. $R < 100$ meters) to the targeted location, then the app sets a proximity alert to detect the timestamp of entering to and leaving from the coffee shop. However, if the user is not close enough ($R >= 100$ meters), then the app schedules an alarm to check the location again on the estimated arrival time of the user. The estimated arrival time is calculated based on the user's current speed and her distance to the coffee shop. Before scheduling further alerts, our app expects user to make some progress towards the coffee shop. If the user does not make any progress towards the targeted location n consequtive times, then the job is cancelled. If there is a progress towards the targeted location, then the proximity alert lets our app know at time t_1 where user enters to and at time t_2 where user leaves from the coffee shop. Having these two timestamps, wait-time is just the difference of $t_2 - t_1 - \epsilon$ where ϵ is the mean error which is accounted for the accuracy of this method (around 1.5 minutes).

Location provider selection in smartphones:
Smartphones provide various ways to obtain user's location. GPS, cell towers, and WAPs are the main ones. Aside from these, there are two other mechanism to learn location: last-known location and passive location learning—a way of getting location whenever another application requests it. The location provided by these methods has its accuracy and timestamp on it. In Android, an application for tracking the location is free to choose among these methods. However, given the different sensing costs of these methods in terms of energy and time, we use a dynamic and adaptive selection of the location providers.

Since we are interested in the user's proximity to the coffee shop, distance is an important parameter for our system. Hence, using the user's distance from the targeted location, our app dynamically selects to use cell tower, WAP or GPS as a provider (see Figure 4). Android combines and uses cell tower and WAP locations as the network location and differentiate it from the GPS location. In Android, our app first looks for the last-known location and calculate the user's distance by taking accuracy and timestamp of the location into account. If the Coffee Shop is very far away or the location is recent and accurate, then we use the last-known location. Otherwise, we take the network location which in general has a better accuracy. If it doesn't satisfy the requirements too, then we take a more conservative approach and learn the location from GPS which in general has a good accuracy of 10 meters. Note that last-known location and network location is mostly available and they are not costly in terms of the battery. On the other hand GPS is costly and can easily drain the device's battery if over-used.

Wi-Fi sensing Our second method for wait-time detection is to leverage the WAP in the coffee shop. Nowadays, most of the coffee shops have Wi-Fi to provide their customers easy and fast access to internet. Moreover, these WAPs generate beacons to broadcast their existence within the radius of 50-100 meters. And each beacon associates with a unique BSSID.

Once our system has a hint that the user will potentially visit the coffee shop, it then starts to monitor Wi-Fi beacons periodically to detect entering to and exiting from the targeted location. With the help of scanning ability of WAPs without connecting to them (provided on the Android platform), our system tracks Wi-Fi beacons easily with little energy consumption. Having the Wi-Fi scan results available, wait-time calculation is just the process of taking difference of $t_2 - t_1 - \epsilon$ where t_1 is the time we start and t_2 is the time when we stop to see Wi-Fi beacons of the WAP. Note that ϵ is the mean error for this method (around 1 minutes, accounting for the scanning period).

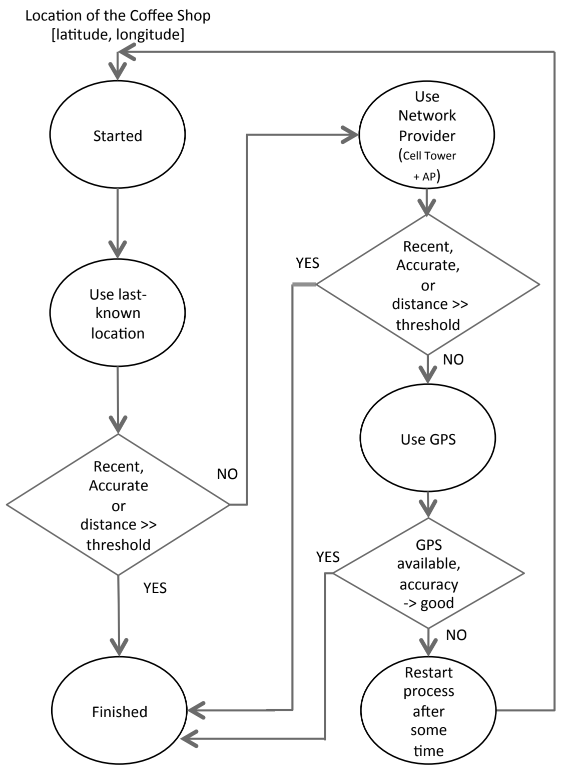
Location of the Coffee Shop
[latitude, longitude]
Started
Use last-known location
Recent, Accurate or distance >> threshold
NO
YES
Finished
Use Network Provider (Cell Tower + AP)
Recent, Accurate, or distance >> threshold
YES
NO
Use GPS
GPS available, accuracy -> good
YES
NO
Restart process after some time

Fig. 4. Selecting among location providers

4.3 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. Since the data is reasonably small, 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. 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.

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 forecaster. The web service serves as the interface between smartphones and the back-end. It accepts wait-times collected from the smartphones and provides wait-time estimations for the querying smartphones. Data collected from web service is fed into the pre-processing module which is responsible mainly for removing outliers and smoothing the data. After pre-processing, model builder builds a model from all the collected data. Lastly, the wait-time forecaster module uses the model and estimates the future wait-times. Below we describe the data that we use for analysis, then we explain wait-time estimation in details.

Data: For our analysis, we used the 8 weeks of collected data (CD) between 2012-02-27 and 2012-04-29 from 8am to 5pm [2]. CD consists of the wait-time detections that the LK app on the smartphones generated and transferred to the back-end. This data is non-uniformly distributed along time, and is initially sparse over the period it has been collected. The sparseness of the data gradually decreases as the popularity of the application increases.

The raw CD contains outliers due to false-positives: some customers sit in the coffee shop after being served, and some leave the coffee shop without being served. Thus, CD needs smoothing and outlier removal methods to filter out the samples that do not provide direct information of wait-times. We utilize distance-based outlier detection method defined in [14] to enable more accurate modeling. Also, during preprocessing, we remove wait-times that are smaller than 2 minutes (which is less than the observed min. service time) and larger than 20 minutes (which is larger than the observed max. wait-time). As a result there are total of 1782 data points in our CD dataset.

Separate from CD, we also manually collected one week of Observed Data (OD) by physically observing and noting the wait-time in the coffee shop. OD consists of wait-time data collected with equal intervals (every 10 minutes) between 8am-5pm, and provides an accurate state of the line wait-times (without false-positives) for that week. Of course it is tedious to collect OD, and we cannot expect to obtain OD for all the businesses added to LK. Since we want our

[2] A week is excluded due to spring break

LK service to be scalable and bootstrap itself from the beginning, the wait-time estimation techniques we developed do not rely on OD. We use OD only to observe how wait-times changes along the time and to extract max. and min. wait-times.

Wait-time estimation problem: The problem is to estimate the line wait-time for any arriving query by using CD. 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 estimation has been usually based on regular uniform time-series that contain enough samples. 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. To achieve this, we developed two estimation approaches. Our two estimation approaches represent a spectrum from a fast heuristic to a time-series model. Both approaches are designed to handle insufficient data and adapt/improve as more data is collected.

Our first approach is a Nearest neighbor estimation (NNE) based on constrained nearest-neighbor search in a multi-dimensional space. This approach is dynamic and works well with non-uniform and sparse CD. In the second approach, we improve NNE by building time-series model on CD using the previous history of wait-times. We show that both approaches provide considerably accurate estimations.

In Section 5.1 we explain our nearest neighbor estimation technique. In Section 5.2, we present the model-based time series estimation and finally in Section 5.3, we compare the estimation (forecasting) capabilities of the mechanisms.

Evaluation: We evaluate the approaches using their resulting Mean Absolute Error (MAE). Given a set of n wait-times: $y_1, y_2, ..., y_n$ and their estimated values: $f_1, f_2, ..., f_n$, MAE is defined in Equation 1.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |f_i - y_i| \tag{1}$$

5.1 NNE: Nearest neighbor estimation

The main idea in this method is to predict the queried values using the previous history of wait-times based on their similarity of values. To this end we identify k nearest neighbor points (k-NN) for the query, where similarity is defined with respect to the estimation potential. The key here is to design a similarity (neighborness) function that optimizes 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, $[w, d, i]$. Each data is associated by a vector $[w_i, d_i, i_i]$, 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], i_i stands for interval of the day and is from the domain [1,54] (there are 54 intervals of 10 minutes between 8am and 5pm). We use weighted L_{ij} to denote the dissimilarity measure between two vectors and define it as the weighted sum of the absolute differences between each dimension;

$$L_{ij} = \alpha(|w_i - w_j|) + \beta(|d_i - d_j|) + \gamma(|i_i - i_j|) \tag{2}$$

Hence, the problem is deduced to find the optimal values for α, β and γ. Below we explain our regression-based optimization method to optimize these values.

Regression-based optimization 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 ($[w_i, d_i, i_i]$). Therefore, we first assume that wait-time is linearly dependent to each dimension of the data vector as in Equation 3. And then we utilize the labeled data points (previous history of wait-times) and assign the weights that optimize the regression function for the labeled data.

$$v_i = \alpha w_i + \beta d_i + \gamma i_i \tag{3}$$

We use linear regression to optimize these weights. Table 1 shows the normalized dissimilarity weights for our 8 weeks of data. Results indicate that, for an estimation interval, the nearest intervals and the intervals from the nearest days have higher similarity than the intervals from previous weeks. This roughly means that the importance of the previous week's data decreases as the time passes. This provides a dynamic way of selecting closest week's data for our method.

Week (α)	**Day (β)**	**Interval (γ)**
0.991	0.130	0.032

Table 1. Weights for dissimilarity measurement using regression

K-NN Estimation After finding the weights (α, β, γ), similar to the k-nearest neighbor algorithm in machine learning [6], our aim is 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. And then, we find the minimum distanced $k = 5$ data points and calculate the average of their wait-times as the estimated value. First row of Table 2 shows the modeling error of NNE. Note that results are in terms of seconds. Same applies to all following results.

Model	**MAE**
NNE	234
Exp. Smoothing	34
Holt Winters	55

Table 2. Modeling error for the models

5.2 Model-based estimation

In this section, we explain how we apply time series theory to wait-time estimation problem as an improvement to our NNE method. As we stated earlier, time series analysis usually depends on uniform and equally spaced data. However, CD does not fully convey these features. Besides it has outliers that makes modeling more difficult. In this section, we present solutions to overcome these shortcomings on CD. We first present how we generate uniform, equally-spaced time series data which we call enhanced collected data (e-CD). Second, we provide the analysis of e-CD. Finally, we fit the data to the time series forecasting models and provide results to evaluate their performance while comparing with our previous approach.

Missing data problem on CD In a typical data collection process, missing data can occur for a variety of reasons including system failures, communication failures etc. In our case, we have missing data because in some intervals either there was no users using our app in the coffee shop or the existing users were unable to upload detected wait-times to our server yet. Although, we expect these behaviors to minimize as more users use our app, until then we need to handle these missing data and provide accurate wait-time estimation.

There are variety of ways to handle missing data; imputation, partial deletion, interpolation and regression are just some of the methods [18]. We adopt a regression-based estimation for filling and constructing equally-spaced (10 minutes of intervals) data. Specifically, we adopt our nearest neighbor estimation (NNE) method that we defined in the previous section for handling missing data. Since we cover most of the details of the method in previous section, we will not repeat the same phenomena here. Basically, we fill the missing interval using the resulting regression model which is constructed by using previous history of wait-times up to the queried interval. For the intervals that already have data, we simply take the average of the collected data as the representative of that interval. In order to deal with outliers we apply a two-sided moving average smoothing to the data. After these processes we finally constructed equally-spaced (10 minutes), uniform time series; e-CD.

Analysis of e-CD In this section we analyze the enhanced collected data (e-CD) in details. This would enlighten our selection of forecasting models in the next section. Figure 5(a) and (b) illustrate some statistical diagnostics of time series. The bell-shaped histogram plot and the wait-times being close to the straight line in the normal probability plot (although a little bit skewed left for the higher values) indicate that the wait-times follow a normal distribution. Note that time-series models are more suitable for normally distributed data and e-CD exhibits this behavior [3].

Traditionally, time-series data considered to be composed of trend, seasonality, cyclical fluctuation and the random effect. Figure 5(c) and (d) shows trend and seasonal components of e-CD. As can be seen from the Figure 5(c), there is an increasing trend for the wait-time which we believe due to the approaching summer. Figure 5(d) shows the seasonality component of e-CD. It clearly exhibits seasonality and validate the selection of a model which has seasonality component.

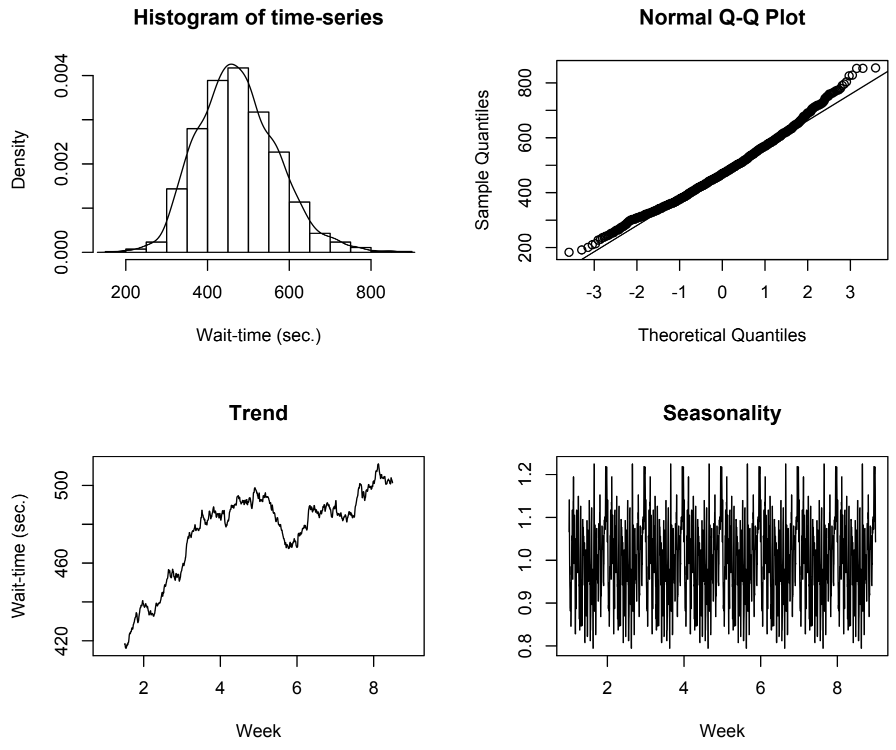

Fig. 5. Statistical plots of e-CD. (a) Histogram of wait-times (b) Normal Q-Q Plot (c) Trend component of time series (d) Seasonality component of time series

Modeling e-CD As explained in the previous section, e-CD exhibits trend and seasonality which motivate us to use a model which takes these into accounts. In addition, it should be as light-weight as possible and can be incrementally updated as more data comes. This encourages us to use Holt-Winters forecasting method which is a widely used time-series forecasting model based on exponential smoothing [10]. We experimentally select multiplicative seasonal model and choose data(α_{hw}), trend(β_{hw}) and seasonal(γ_{hw}) smoothing factors dynamically as more data comes. We've also considered other forecasting methods such as ARIMA (Autoregressive integrated moving average) and exponential smoothing. However, we observed that ARIMA incurs significant computational costs, and therefore we opt-out it and compare Holt Winters with the ad hoc exponential smoothing. Second and third rows of Table 2 shows the modeling error of Holt Winters and exponential smoothing. It is clear that the both models fits perfectly to our e-CD in comparison to our initial NNE method.

5.3 Comparison of Models

In this section, we compare the models built in the previous two sections in terms of their forecasting capabilities. Table 3 shows the forecasting errors of the models for the last two weeks of our experiment. It is expected that forecasting error will be higher than the modeling error as the modeling takes all data into account from the beginning. On the other hand, forecasting only uses the previous data from the queried one. As shown in the Table 3 Holt Winters outperforms NNE in a significant scale. Figure 6 shows the weekly MAE for each method. Holt Winters and exponential smoothing outscores the NNE for all of the weeks consistently. We believe this is due to the fact that wait-time changes steadily rather than immediately over time and both Holt Winters and exponential smoothing account this fact in its formulation by steadily changing the estimation with new incoming data. Moreover, Holt Winters and exponential smoothing exhibits similar errors where Holt Winter beats exponential smoothing in small margins for most of the weeks. We believe that margin will increase as more data comes and as the data exhibits more trend and seasonality. Figure 7 shows the collected data (including replaced missing values) and forecasted values for the last two days of the last week of our experiment for Holt Winters method. Forecasted values nicely fitted to the actual values. Our current wait-time estimation model based on Holt Winters and updated as more data accumulates.

Model	**MAE**
NNE	227
Exp. Smoothing	156
Holt Winters	155

Table 3. Forecasting error for the last two weeks

6 Deployment

Section 5 presented analysis and experiments of the collected data for building an accurate wait-time estimation model. In this section we present other information from our deployment. We implemented native Android and iPhone apps for LK

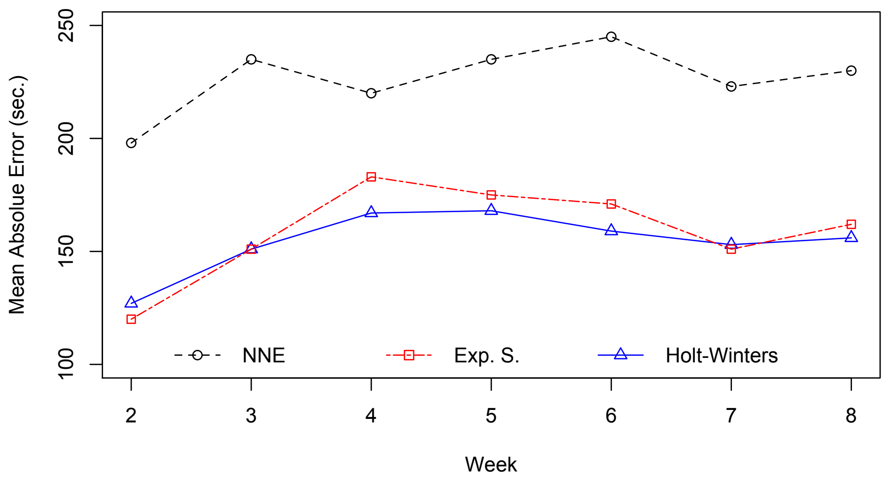

Fig. 6. Weekly forecasting errors for models

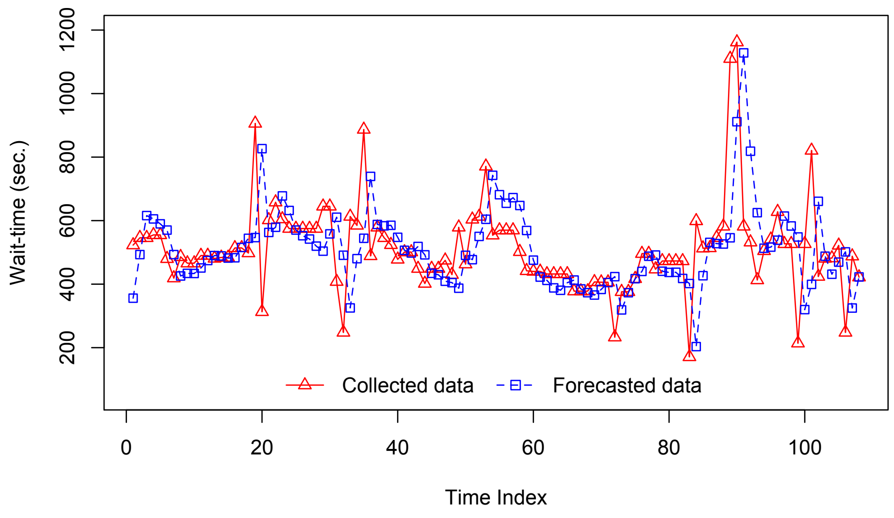

Fig. 7. Collected vs. forecasted data for the last two days of the last week for Holt Winters method. Time Index spans 10 minutes of interval between 8am and 5pm for two days.

and made this available at the corresponding appstores for free. While Android app is written in Java using Eclipse and Android SDK, iPhone app is written in Objective-C using Xcode and iOS SDK. For the sake of scalability, we hosted back-end at AWS EC2. A screenshot of the application is shown in Figure 8. It shows the current wait-time and a graph showcasing the past (left) and future (right) estimates of wait-times. We advertised LK through handing out fliers and putting ads on Facebook pages of various Student Clubs. As of writing this paper, LK was downloaded by more than 1000 users in our campus. We received lots of positive feedbacks about the accuracy of estimation from the users.

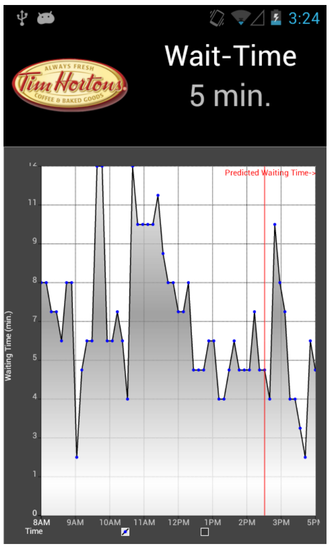

Fig. 8. A screenshot from Android app

Since the iOS platform does not provide a lot of development flexibility, we were unable to implement Wi-Fi sensing for wait-time detection and ended-up using only the location-sensing based solution on the iPhone platform. This was not an issue for the Android platform and both Wi-Fi sensing and location-sensing solutions are fully implemented on the Android. On the other hand, we found that the iOS platform had its own advantages: it was easier to implement a robust location-sensing on the iPhone than on the Android platform.

As we explained in Section 4, LK receives wait-time detection from two sources of information, i.e. location and WAP. Our 8 weeks of CD show that 65% of incoming data is received from location-sensing (iPhone + Android), and the remaining 35% is received from Wi-Fi based sensing (only Android). Figure 9 shows the number of data points for the weeks we used for analysis, except that data is not preprocessed and it is for all day.

7 Discussion

While we presented LK's deployment for one coffee shop, we believe LK's deployment can be extended for other coffee-shops and businesses such as Post Offices, Banks and DMVs. To add a new business to the LK, we only require

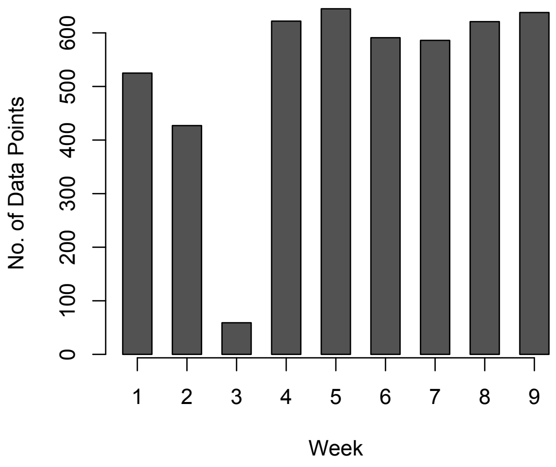

Fig. 9. Incoming number of data points (for all day). Week 3 is spring break and excluded from the wait-time analysis.

the geographical locations, i.e. latitude and longitude, of the business. After a business is added, LK immediately starts receiving line 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 times for a week, or offer promotions and coupons for users who install the LK app and check-in frequently. Below, we include more details on how to scale LK to a large set of locations.

7.1 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. During 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.

7.2 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 [8] 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 Places.

7.3 Improving wait-time detection

As we explained in previous sections, wait-time detection component of LK cannot differentiate between seated customers and the customers waiting in line. In our deployment, majority of the customers leave the coffee shop immediately, therefore, false-positives do not constitute a problem. However, until enough user base is formed, it is possible that wait-time estimation at new businesses might suffer from these false-positives. In order to eliminate these false-positives in wait-time detection, we will try to distinguish between seated customers and waiting customers by employing the state-of-the-art activity recognition techniques [15, 13]. These techniques use the accelerometers in the smartphones to differentiate between different behaviors, including sitting and standing.

8 Conclusion

We described the design, implementation and deployment of LK, a crowdsourced line wait-time monitoring service. LK consists of two main parts: smartphone and cloud back-end components. Smartphone component provides automatic, energy efficient and accurate wait-time detection by using domain specific optimizations for both Android and iOS. And cloud back-end provides accurate wait-time estimation based on collected data from smartphones. In wait-time estimation, we introduced a novel solution based on a constrained nearest-neighbor search in a multi-dimensional space. We then improve it by adapting two time-series forecasting methods namely exponential smoothing and Holt Winters. Our experiments show that, we managed to reduce the mean absolute error of our service to be less than 2-3 minutes. In our future work, we will add new businesses to LK and try to scale our wait-time estimation service to a nationwide deployment.

References

1. Android SDK, `http://developer.android.com`.
2. Xuan Bao and Romit Roy Choudhury. Movi: mobile phone based video highlights via collaborative sensing. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 357–370, New York, NY, USA, 2010. ACM.
3. Peter J Brockwell and Richard A Davis. *Time series: theory and methods.* Springer-Verlag New York, Inc., New York, NY, USA, 1986.

4. Ionut Constandache, Xuan Bao, Martin Azizyan, and Romit Roy Choudhury. Did you see bob?: human localization using mobile phones. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, pages 149–160, New York, NY, USA, 2010. ACM.
5. R. B. Cooper. *Introduction to Queueing Theory.* North-Holland, New York, NY, second edition, 1981.
6. B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA, 1991.
7. Christos Faloutsos. Mining time series data. In *SBBD*, pages 4–5, 2005.
8. Foursquare venues platform, `https://developer.foursquare.com/overview/venues`.
9. Raghu K. Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: Current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.
10. Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.
11. iOS SDK, `https://developer.apple.com`.
12. Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. Distance measures for effective clustering of arima time-series. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 273–280, Washington, DC, USA, 2001. IEEE Computer Society.
13. Matthew Keally, Gang Zhou, Guoliang Xing, Jianxin Wu, and Andrew Pyles. Pbn: towards practical activity recognition using smartphone-based body sensor networks. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 246–259, New York, NY, USA, 2011. ACM.
14. Edwin M. Knorr and Raymond T. Ng. A unified approach for mining outliers. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '97, pages 11–. IBM Press, 1997.
15. Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, March 2011.
16. Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. A survey of mobile phone sensing. *Comm. Mag.*, 48:140–150, September 2010.
17. D V Lindley. The theory of queues with a single server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(02):277, 1952.
18. Roderick J A Little and Donald B Rubin. *Statistical analysis with missing data.* John Wiley & Sons, Inc., New York, NY, USA, 1986.
19. Oded Maimon and Lior Rokach, editors. *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Springer, 2010.
20. Emiliano Miluzzo, Nicholas D. Lane, Shane B. Eisenman, and Andrew T. Campbell. Cenceme: injecting sensing presence into social networking applications. In *Proceedings of the 2nd European conference on Smart sensing and context*, EuroSSC'07, pages 1–28, Berlin, Heidelberg, 2007. Springer-Verlag.
21. Chuan Qin, Xuan Bao, Romit Roy Choudhury, and Srihari Nelakuditi. Tagsense: a smartphone-based approach to automatic image tagging. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 1–14, New York, NY, USA, 2011. ACM.
22. Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 315–330, 2010.