# PocketParker: Pocketsourcing Parking Lot Availability

Anandatirtha Nandugudi, Taeyeon Ki, Carl Nuessle, and Geoffrey Challen
University at Buffalo
{ans25,tki,carlnues,challen}@buffalo.edu

## Abstract

Searching for parking spots wastes time while generating frustration and pollution. To address these parking problems, we present *PocketParker*, a crowdsourcing system that uses smartphones to predict parking lot availability. We consider PocketParker to be an example of a subset of crowdsourcing we call *pocketsourcing*. Pocketsourcing applications require no explicit user input or additional infrastructure, running effectively without the phone leaving the user's pocket. Users interact with PocketParker only when looking for parking spots. PocketParker detects arrival and departure events by leveraging existing activity recognition algorithms. Detected events are used to maintain per-lot availability models allows the PocketParker server to respond to client availability queries. By estimating the number of *hidden drivers*—those not using PocketParker—we can use a small fraction of monitored drivers to estimate arrival and departure rates and make accurate predictions. Our evaluation uses multiple data sets to determine the accuracy of each PocketParker component and the system as a whole. We show that PocketParker quickly and correctly detects parking events, and that our availability estimator is accurate and robust to the presence of hidden drivers. Finally, we deploy a prototype and use camera monitoring of several parking lots to demonstrate PocketParker's performance in the wild.

## 1. INTRODUCTION

Parking lots present a difficult search problem. Drivers lack the visibility to determine where spots are available, and may spend a non-trivial amount of time searching for a spot. The problem is difficult enough that WikiHow includes directions [3], and the Wall Street Journal has published an online article [2] with tips on spot stalking for shoppers during the holidays. Searching not only generates frustration but also wastes energy and produces harmful carbon emissions.

Online smartphone application stores such as Google Play and the App Store are teeming with apps claiming to help you find a parking spot. Although some drivers may find these applications useful, they either do not provide real-time parking lot availability or simply display publicly-available information. Several research projects have attempted to address these limitations [9, 10, 13, 15, 16], but include requirements rendering them impractical, such as additional infrastructure [15], on-vehicle equipment [16] or vehicular networking [13, 16], or onerous manual user input [10]. In contrast, we believe the solution is already in your pocket.

We present *PocketParker*, a system that predicts parking lot availability using smartphones. Unlike previous approaches, PocketParker requires no additional infrastructure, no vehicle modifications, and no user input, only installation on a small percentage of the 100 million smartphones already in use in the US [1]. PocketParker runs unattended in the background and uses the accelerometer to detect parking lot arrivals and departures. These are forwarded to a central server, which incorporates them into per-lot availability models. This allows PocketParker to order lots accurately by the probability that they contain an available spot. In general, we consider our approach to be an example of a subset of crowdsourcing that does not require any manual user input, which we call *pocketsourcing*.

Providing parking availability predictions requires efficiently and accurately detecting parking-related events, and incorporating the effect of *hidden drivers*—those not using PocketParker—into our availability model. We address the first challenge by designing a simple yet effective event detector which uses the smartphone accelerometer to efficiently detect arrival and departure events, triggering energy-hungry GPS acquisition only when necessary. We address the second challenge by designing an availability estimator that maintains a probability model for each lot continuously incorporating data from PocketParker clients. We use detected events both to estimate arrival and departure rates and to make changes in real time. Part of the key to our approach is the observation that even with limited information, there are moments when PocketParker can be certain about the availability of a parking spot in a given lot, and this certainty allows PocketParker to

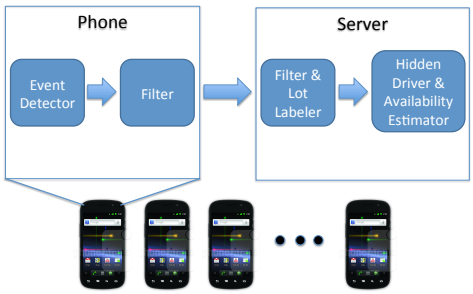



Figure 1: **The PocketParker architecture.** Events generated by an activity detector running quietly on each smartphone are processed by a central server and used to estimate parking lot availability.

assist users.

We perform a careful evaluation of PocketParker using a variety of methods tailored to each system component. We evaluate our parking event detector in a controlled environment with eight volunteers participating in ten parking scenarios. We design a simulator to evaluate our parking availability estimator, which gives us the flexibility to experiment with a variety of parameters and parking lot types. Finally, we evaluate the overall effectiveness of PocketParker by deploying it with 105 smartphones used by our participants over forty five days. To obtain ground truth, we deploy four cameras that monitor two parking lots over two weeks. We inspect and hand-code four days' worth of images of these lots to measure their true availability. Altogether, our results show the efficiency and accuracy of PocketParker.

As depicted in Figure 1, PocketParker has several components distributed across participating smartphones and a backend server. The rest of our paper describes each component in detail. We start by presenting related work in Section 2 in order to distinguish PocketParker from multiple previous efforts at parking monitoring. Next, in Sections 3 and 4 we describe two major components of PocketParker: our parking event detector and availability model. We base our evaluation in Section 5 on simulations and controlled exeriments stemming from two real-world deployments. Finally, we discuss limitations and future work in Section 6 before concluding in Section 7.

## 2. RELATED WORK

Many previous projects have addressed components of our problem, including activity detection, parking lot monitoring using various types of infrastructure and additional sensors, and transportation-related tracking. Below we discuss related efforts and their relevance to PocketParker.

### 2.1 Activity Detection

Activity detection is a basic primitive that enables other higher-level functionalities. Several previous systems [12, 14, 18, 23, 21] have proposed techniques for activity detection, solving the main challenges of energy efficiency and accuracy. Yang et al. [23] develop an algorithm that detects if a driver is using a phone by sending high-frequency beeps via in-car speakers. EEMSS [21] is a system that provides continuous identification of general human states such as walking, driving, and being in an office. Reddy et al. [18] propose a classification approach that determines human movement states such as walking, running, biking, or vehicle-traveling. Constandache et al. [12] use smartphone accelerometers to determine users' walking trails. Yan et al. [22] propose an adaptive approach that dynamically adjusts the accelerometer sampling frequency for conserving energy. Keally et al. [14] use a combination of on-body wireless sensors and smartphones to classify human states. Our system can benefit from these techniques for detecting parking-related events; however, we find it sufficient to use a simple detector since it avoids the complexity of detecting events unrelated to parking.

### 2.2 Parking Lot Monitoring

There are a large number of parking lot applications available in the online smartphone application stores. A typical parking lot application provides location as well as pricing information and allow reservation of a parking spot. Some applications also report parking lot availability based on publicly available information. To the best of our knowledge, there is no application that automatically infers parking lot availability by monitoring drivers.

Most close to our work is ParkNet [16], a system that estimates street parking availability. ParkNet uses vehicles equipped with GPS and a ultrasonic range finder that scan the surrounding areas and detect empty street parking spots. Compared to ParkNet, our approach relies only on smartphones and do not require any additional input.

A few systems have been proposed to assist parking via vehicular ad-hoc networks or crowdsourcing. Delot et al. [13] propose a parking lot reservation system in a vehicular network. Chen et al. [10] proposes a crowdsourcing approach that asks participants to report their surrounding's parking availability. Caliskan et al. [9] propose an availability prediction model based on information exchanged by vehicles in a vehicular ad-hoc network. Their approach assumes that, for each parking lot, vehicles that drive by the parking lot receive the parking lot information such as the capacity, the occupancy, the arrival rate, and the departure rate. Since the propagation delay in a vehicular network makes this information stale, a prediction model is used to estimate current availability. In contrast to our work, this approach assumes that the necessary information is ini-

tially accurately measured at each parking lot.

## 2.3 Tracking-Related Projects

A few previous systems have investigated techniques for automatic transit tracking [8, 19, 24]. From the high-level point of view, some of the techniques such as activity detection and location tracking that transit tracking requires bear similarities to our techniques; however, these techniques need to be optimized and tailored towards different scenarios, hence the specifics vary widely.

Thiagarajan et al. [19] propose an automatic, real-time transit tracking approach that uses smartphones of public transit riders as data sources. They propose an algorithm to detect when a user is traveling in a vehicle and an algorithm to detect if a vehicle is a public transit vehicle. EasyTracker [8] uses smartphones deployed in buses to enable automatic transit tracking. The goal of EasyTracker is to require no other input than what is from the deployed smartphones. The system combines a few mechanisms to realize this goal such as route extraction, stop extraction, and arrival time prediction. Zhou et al. [24] propose a bus arrival time estimation system based on smartphones used by public transit riders. They combine multiple sources such as accelerometer data, audio, and cell tower signals to detect if a rider is in a public transit vehicle and if so, which bus it is.

Other systems have used smartphone sensors to enable a variety of tracking tasks. VTrack [20] is a traffic monitoring system that combines readings from multiple sensors for travel time estimation. StarTrack [7] provides general abstractions for appliations that need tracking functionalities such as recording, comparing, and querying tracks.

## 2.4 Urban On-Street Parking

A recent academic study also focus on the problem of locating on-street parking in urban areas. Nawaz et al. [17] leverage the ubiquity of WiFi beacons to monitor on-street parking events in a similar fashion to that performed by PocketParker. Our study, borne out of suburban campus locale, must cope with alternative sensing mechanisms in the wake of no proximate WiFi signals. We have also tuned our tracking and reporting methodology to address the needs of lot rather than street parking. Nationally, data suggests that the proportion of spots in lots is anywhere from somewhat less than to five times the number of spots on streets. [11]

# 3. EVENT DETECTOR

The inputs to PocketParker's availability estimation algorithm are arrival and departure events generated by an activity detector running unattended on users' mobile devices. While considerable previous research has explored activity detection using mobile sensing [12, 14, 18, 23, 21], we design a custom parking event detector tailored to the goals of PocketParker. In this section, we briefly describe this detector and other portions of our system that run on the smartphone itself.

## 3.1 Parking Events

PocketParker assumes that transitions between walking and driving that occur in and adjacent to locations known to be parking lots constitute either arrival (driving to walking) or departure (walking to driving) events. We thus must be able to discern between walking and driving states of the user, and to do so fast enough to fix the the location of the parking lot in which the event took place. Detecting these states could be achieved using continuously-sampled GPS data would consume too much energy for an effective pocketsourcing solution. Rather, we rely on duty-cycled accelerometer data to classify the user behavior into one of three states: walking, driving, or idle.

The initial inference of user states yielded by accelerometer sensing is subsequently refined with GPS and WiFi sense data to yield the desired goal: detection of arrival and departure events. The mobile device reports these events, along with their locations, to the server. Before recording the event, the server verifies its location against a pre-compiled list of known parking lot locations. This final step eliminates events that are either incorrect (e.g., a user parking in a field) or unwanted (e.g., a user genuinely parking but in a loading area rather than in a parking lot).

Subsequent to the conclusion of our research, Google released an update to its closed Android binaries that contained user activity detection functionality. As this was similar to that developed for PocketParker, substituting this code would not have materially affected the detection of arrival and departure events. Both algorithms treat user state detection in terms of relative liklihoods. [6] False detection could be further minimized with additional sensing, particularly GPS data, or with more computationally intense detection algorithms. We have deliberately shied away from such an approach however, as the inherent nature of our application dictates that the bulk of testing and filtering takes place on an energy-constrained mobile platform.

# 4. AVAILABILITY ESTIMATION

In order for parking events to be useful, they must be incorporated into a model allowing us to predict parking lot availability. Our goal is to respond to queries with the probability that a given parking has a space available, information that can be used in several ways to determine what lots to search and in what order. PocketParker's estimator uses the events produced by

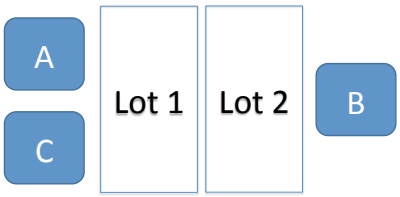



Figure 2: **Example parking lot setup.** Two lots and three destinations are shown.

our parking event detector both to estimate the rates at which drivers are searching and departing from the lot and to adjust the availability probability directly. In this section, we present both the design of the PocketParker client parking lot availability estimator and portions of the backend server for our system.

## 4.1 Overview

Figure 2 shows an example setup with two parking lots and two destinations that are used throughout this section. For each lot PocketParker maintains a time-varying probability that the lot has $n$ free spots $P(t, n)$. While we are mainly interested in the probability that the lot has a space available $P_{free} = \sum_{n>0} P(t, n)$, we maintain separate probabilities for each number of free spots so that we can manipulate individual probabilities in response to events and queries as described below. We bound the count probability distribution to lie between 0 and the capacity of the parking lot. Section 4.2 describes how PocketParker estimates lot capacity.

PocketParker's estimator receives two types of events: arrivals and departures. However, for each arrival in a given lot, a number of additional lots may have been searched unsuccessfully, information critical to the accuracy of our availability model. Section 4.3 describes how PocketParker determines relationships between parking lots, and Section 4.4 describes how we combine that information with arrivals to estimate implicit search behavior.

Between events we want to maintain our availability model by estimating the rate at which departures and searches are taking place. PocketParker must use the events it can detect to estimate the rate at which events are taking place in the lot, which includes the effect of drivers not using PocketParker, which we call *hidden drivers*. Accomplishing this requires that we estimate the ratio between monitored and hidden drivers. We describe an approach to doing so in Section 4.5. With an estimate of the hidden driver ratio, we can scale the search and departure rates accordingly, described in Section 4.6. Finally, Section 4.7 describes how we integrate all of this information to update our availability estimate as arrival and departure events are received.

## 4.2 Estimating Lot Capacity

PocketParker requires an estimate of lot capacity $C$ in several places. First, we use this estimate to bound $P(t)$ such that $P(t, n > C) = 0 \; \forall \; t$. Second, we use the capacity to determine the number of hidden drivers, as detailed in Section 4.5. To calculate a lot capacity, we use the location of the parking lot obtained from the OpenStreetMap database [5]. We derive the lot size from its location and then divide the total size by that of a typical standard parking spot lot design [4]. In comparing this estimate with manually counted capacities for the five lots monitored by our deployment, capacity estimates were in all cases within 6% of their true capacities.

Errors in the capacity can result if the size of parking spots in the lot differ from our estimate, or if the parking lot is not efficiently packed with spots. Given the incentive of parking lot designers to maximize capacity, we believe that the second case will be unlikely. Parking spot sizes, however, may vary significantly from lot to lot or based on the lot's location. To improve our estimate, we may need to incorporate location-specific parking spot size estimates. Alternatively, mapping databases may be directly annotated with the number of spots per lot.

## 4.3 Lot Relationships

PocketParker's detector identifies only arrivals and departures. However, understanding and incorporating search behavior is critical to our model. For example, if we observe the arrival rate fall at a given lot, it may be because the lot is full, or it may be simply because fewer drivers are arriving and the lot still has many spaces available.

In order to estimate search behavior, we need to understand the relationships between parking lots. This requires two additional pieces of data about each lot: one or multiple destinations, and a desirability index. The destination represents the place the user is going when they park in a given lot, and note that some lots may be associated with multiple destinations. In Figure 2, lot 1 may be associated with destinations A, B and C; while lot 2 is only linked to B.

The desirability index produces an ordering of lots associated with a given point-of-interest based on how preferable they are compared with other lots. We assume that most users will park in desirable lots if they are available, and may have searched in more desirable lots before parking in a lot desirable lot. In Figure 2, if Lot 2 is associated with destination A it will probably receive a lower desirability score than Lot 1 because it is further away.

While this information is not currently part of open mapping databases, we believe that it is straightforward to collect. Parking lot operators and business owners can annotate the mapping database with destinations

for each lot. In addition, data from navigation tools may be able to automatically link destinations with lots by noting where users park after requesting directions to a particular place. The desirability index may also be determined by navigation tools observing what lots are searched by users on their way to a particular destination. Lacking these traces, simple proximity to the destination may determine the desirability index directly. As example of this automatic annotation, in Figure 2 if both lot 1 and 2 are associated A, we consider lot 2 less desirable because lot 1 lies between it and the destination.

## 4.4 Implicit Searches

With an understanding of lot relationships we can use observed arrivals to model implicit—or unobserved—searches. When a user parks in a given lot, we use the desirability index of the lot to add unsuccessful searches in more desirable lots associated with the some destination. There are two challenges to this approach. First, as described above, lots may be associated with multiple destinations. Second, the user may not have actually performed the search. After discussing both of these issues below, Section 4.7 describe below how PocketParker incorporates the information from implicit searches in a way sensitive to these uncertainties.

### *4.4.1 Determining the destination*

If a lot is associated with multiple destinations, we cannot uniquely determine the destination of the user. However, this only becomes important if the two destinations would produce different desirability rankings for affected lots. For example, in Figure 2, if lots 1 and 2 are both associated with destinations A and C, but not with B, then an arrival with an unknown destination into lot 2 can always be used to generate an implicit search in lot 1, since the desirability ranking for the two lots are unchanged if the destination is either A or C. However, if both lots 1 and 2 are associated with all three destinations, then an arrival detected in lot 2 becomes more ambiguous. If the user was trying to go to destination A, it may mean that lot 1 was searched and is full; however, if they were trying to go to destination B, it may not indicate anything about lot 1.

If lot destination annotations are generated by mapping software, we can use this data to estimate the probability that a user is going to each of the destinations associated with a particular lot. Instead of generating a single implicit search in one lot, we generate multiple implicit searches in each of the lots weighted by the destination probabilities. Lacking these probabilities, we simply generate implicit searches in each destination associated with a given lot.

### *4.4.2 Speculative searches*

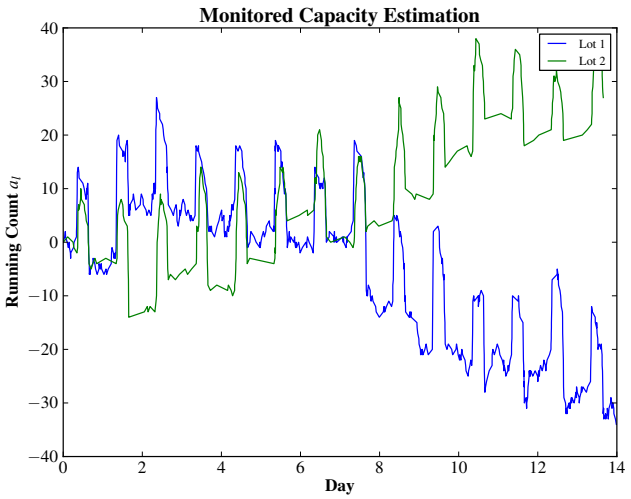



Figure 3: **Example of capacity estimation.** Running counts for two lots are shown.

If we do not directly observe a user searching a lot before we detect an arrival, we cannot be certain that they performed the search. If the unsearched but preferable lot was available, they may not have searched it because they preferred to choose the first available spot, enjoyed the exercise of walking farther to their destination, or wanted to irritate their passengers. However, these are not the type of users we believe would benefit from or use our PocketParker application, since finding a non-optimal parking spot is fairly simple in most cases.

A more interesting case is where a user has not performed a search before parking in a less-desirable lot because they *believe* the more desirable lot to be full. Users that park regularly at the same destination usually have their own mental models for the availability of spots in certain lots, causing them to discard those lots without searching them if they believe the probability of finding a spot in the desirable lot is low. While this behavior can cause users to miss available spots, these speculative searches are useful inputs since they reflect lots users think are full.

A final corner case that PocketParker does not handle is if all of the lots for a given destination are full, and many undetected unsuccessful searches are taking place. On one hand, if all lots are full then the availability of spots is determined by departures, not arrivals, and so search data is useless anyway. On the other hand, we would still like to identify this situation for users that would prefer to avoid destinations where it is difficult to park. While discussing future work in Section 6, we point out how integrating PocketParker into existing navigation applications could address this problem by making searches explicit, rather than implicit.

## 4.5 Hidden Driver Estimation

Monitored PocketParker users compete for parking

spaces with unmonitored users, which we call *hidden drivers*. While we assume that PocketParker users are generally representative of the entire driving population, we do not assume that all or even a large fraction of drivers will download and install PocketParker. We want our system still to provide accurate predictions with the limited information caused by hidden drivers. To accomplish this, PocketParker needs to estimate the percentage of drivers that are monitored, which we call the *monitored fraction* $f_m$. A low monitored fraction indicates that few users are using PocketParker, whereas a high monitored fraction means most are. Put another way, the amount of uncertainty PocketParker faces when predicting availability is inversely-proportional to the monitored fraction.

#### 4.5.1 *Importance of monitored fraction estimation*

Two examples will illustrate why we need this information and how it is used. First, when a monitored driver leaves a parking lot, the monitored fraction determines how long PocketParker will predict that a spot in that lot is available. As the monitored fraction increases, the probability of PocketParker seeing the arrival into the lot that occupies that spot increases, and we can increase the amount of time that we estimate a spot is available. On the other hand, as the monitored fraction decreases we see fewer arrivals and are faced with more uncertainty. Hence, PocketParker reduces the amount of time it predicts the spot is available. In Section 4.7 we describe how hidden drivers influence the changes to the availability model made when arrivals and departures are detected.

Second, PocketParker uses the arrival and departure rates of monitored drivers to estimate changes to parking lot availability over time. Here we must scale the observed number of events to the actual number of events, which requires an estimate of the monitored fraction. Section 4.6 describes how the monitored rate is used for rate estimation and scaling.

#### 4.5.2 *Estimating the monitored fraction*

PocketParker estimates the monitored fraction by first determining the monitored capacity—the capacity of the lot measured by monitored drivers—and then using our estimate of the lot capacity discussed in Section 4.2. Specifically, given a lot with capacity $C$, the monitored fraction can be estimated as $f_m = \frac{C_m}{C}$. Our task then becomes estimating the monitored capacity $C_m$. To estimate the monitored capacity we maintain a running count $a$ for each lot, decremented when drivers arrive and incremented when they leave. We can consider $a$ as a estimate of the number of spots available in the lot scaled by $f_m$, although we do not bound $a$ to be below the lot capacity or greater than zero.

Figure 3 shows an example of the running count for

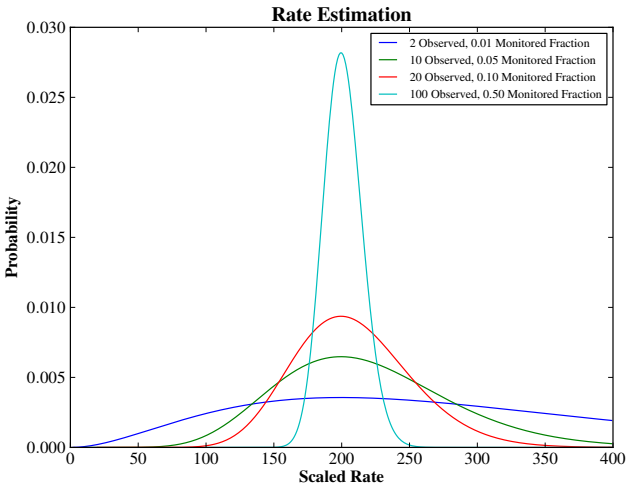



Figure 4: **Example of rate estimation.** Spread of each distribution shows the effect of the monitored fraction on rate certainty.

two related lots over seven days using data generated by our lot simulator described in Section 5.2. Both lots have capacity 200 and the actual monitored fraction is 0.1. As the data shows, the running count experiences long-period (greater than one day) fluctuations due to events missed by our event detector and the randomness associated with the small percentage of drivers being monitored. However, the data also contains short-period (less than one day) fluctuations caused by the dynamics of the lot being monitored, and these fluctuations are roughly the size of the monitored capacity $C_m$, which in this case is 20 spots.

This observation motivates the design of our monitored capacity estimator. First, we bin the data into 24 hour intervals. Next, we identify the largest availability swing over each window. Finally, we average multiple swings together for a period of days to determine the final estimate. This simple approach works well on lots that fill on a regular basis. For the example in Figure 3, our estimator estimates the monitored capacity of lots 1 and 2 as 21.01 and 21.08, respectively, within 10% of the true value in both cases.We perform a further evaluation of our capacity estimator using multiple lot simulations in Section 5.

For lots that do not fill, or do not fill regularly, we may need to produce a weighted sum where larger swings are weighted more heavily given our assumption that they more accurately measure the true monitored capacity of the lot. Another approach is to use the monitored fraction estimated at desirable lots for a given destination, which are more likely to fill completely and often, to estimate the monitored fraction for other less-desirable lots. Here we are making the reasonable assumption that lots connected to the same destination share sim-

ilar fractions of PocketParker users. Finally, PocketParker's monitored fraction estimator runs periodically to incorporate changes in the monitored fraction caused by increasing use of PocketParker.

## 4.6 Rate Estimation

When PocketParker receives arrival and departure event information, it knows something concrete about the state of the lot. However, to predict availability at other times we need to adjust our estimation based on recently-observed events, which we call rate estimation. To estimate the rate of events in the entire population including hidden drivers, PocketParker must scale its rate of parking events by monitored drivers appropriately. Next, we use these scaled estimates to adjust the probability that a given lot has a certain number of spots and has spots available.

During a time interval $t_0$ to $t_1$, PocketParker will observe some number of searches $s_{obs}(t_0, t_1)$ or departures $d_{obs}(t_0, t_1)$ in any given lot[1]. Note that the search count includes both arrivals—successful searches—and implicit unsuccessful searches derived from arrivals at related lots as explained above. However, depending on the monitored fraction $f_m$ the true count $s_{true}(t_0, t_1)$ is likely to be much larger. Rather than simply scaling the count by $\frac{1}{f_m}$, we want to determine the probability distribution over all possible true counts given the rate we observed and the estimated monitored fraction as derived in Section 4.5.2. One reason we do not simply scale by $\frac{1}{f_m}$ is that our uncertainty about the true count should be affected by $f_m$. If all drivers use PocketParker, we know the true count exactly; if few do, we should be uncertain.

To compute the probability distribution we treat $s_{obs}$ as the output of a binomial distribution with probability $f_m$ and vary the number of trials. Specifically:

$$P(s_{true}|s_{obs}) = C \cdot \binom{s_{obs}}{s_{true}} f_m^{(s_{obs})} \cdot (1 - f_m)^{(s_{true} - s_{obs})} \quad (1)$$

where $C$ is a renormalization constant equal to $\sum_{s_{true}} P$. Figure 4 shows the resulting distribution for three different values of $f_m$ with $s_{true} = 200$. While for all four values of $f_m$, 200 is the most likely true count, as we desired the spread of the distribution increases with decreasing $f_m$.

### 4.6.1 Updating the count probabilities

Given the probability that a lot has $n$ free spots at time $t_0$, $P(t_0, n)$, we want to estimate the probabilities $P(t_1, n)$ at a later time $t_1$. PocketParker uses recently-observed arrivals, implicit searches and departures to estimate the search $s_{est}$ and departure $d_{est}$ rates the lot experienced between $t_0$ and $t_1$. Currently, we use arrival and departures over a fixed-size window of time $I$ before $t_0$, $s_{obs}(t_0 - I, t_0)$ scaled to the length of the interval $t_0$ to $t_1$:

$$s_{est}(t_0, t_1) = s_{obs}(t_0 - I, t_0) \cdot \frac{(t_1 - t_0)}{I}$$

The value of $s_{est}(t_0, t_1)$ is then scaled as described above to determine the distribution of $s_{true}$. PocketParker assumes the rates experienced over the last $I$ time interval will continue. It may be possible to perform better rate estimation by using historical information, but this is left as future work.

The distribution of search rates $s_{true}(t_0, t_1)$ represents the probabilities that the number of available spots in the lot will decline, whereas the departure rate $d_{true}(t_0, t_1)$ represents the probability the number of spots will increase due to departures. The convolution of $-1 \cdot s_{true}$ and $d_{true}$, $\Delta(t_0, t_1)$, represents the change in the number of spots produced by the specific combination of arrival and departure rates. A further convolution of $\Delta(t_0, t_1)$ with $P(t_0, n)$ produces $P(t_1, n)$, the desired probability at $t_1$:

$$P(t_1, n) = P(t_0, n) * (-1 \cdot s_{true}(t_0, t_1) * d_{true}(t_0, t_1))$$

where $*$ represents the discrete convolution.

Note that the convolution of $P$ with $\Delta$ can cause non-zero probabilities in $P$ that violate our boundary conditions, namely that $P(n < 0) = 0$ and $P(n > C) = 0$ where $C$ is the estimated capacity of the lot. To correct this, we simply set $P(n = 0) = \sum_{n<0} P(n)$ and $P(n = C) = \sum_{n>C} P(n)$, assigning all the probability that the lot has less that zero free spots to the zero state and all probability that it has more than the capacity of the lot of free spots to the empty state.

### 4.6.2 Rateless spreading

If the departure rate exceeds the arrival rate, the probability mass of $\Delta$ will lie primarily to the positive side and it will shift $P$ in the positive direction, producing higher probabilities that more spots are available in the lot and lowering the probability that the lot is full. The opposite is true when the search rate exceeds the arrival rate.

An important case is intervals during which PocketParker has observed neither arrivals nor departures in a given lot. In this case, $\Delta$ will be centered around 0 but have a spread determined by the monitored fraction. Its effect on $P$ will be to redistribute the probability mass more evenly across the entire interval from 0 to $C$. Taken over many intervals, the probability of the lot having any number of spots available will equalize, which is what we would expect: after a long period without any information, all states become equally likely and we cannot make an accurate prediction of the state of the lot. Note also that the speed at which the probabilities are redistributed through rateless spread-

[1]Without loss of generality our examples of scaling and estimating rates use notation for the search rate.

ing is determined again by the monitored fraction. The fewer drivers we monitor, the more quickly we lose all memory of the state of the lot.

## 4.7 Online Updates

Finally, we conclude by describing how PocketParker uses arrival to adjust its availability model instantaneously at runtime. Each arrival and departure received at time $t$ represent strong positive information—moments when PocketParker knows either that a spot just existed (arrival) or now exists (departure). PocketParker uses these events to adjust the probability distribution and incorporate this new information. Figure 5 displays an example of the effects of departures, arrivals, and searches discussed below.

Arrivals provide two somewhat conflicting pieces of information. First, PocketParker knows that at the time of the arrival there was a spot free, so in this way arrivals indicate that the lot is not full. However, PocketParker also knows that immediately after an arrival the lot has one fewer available spots. So we incorporate arrivals in two steps. First, we set $P(t,0) = 0$ indicating the availability of a spot and renormalize the distribution. Second, we shift the entire distribution downward by one spot, $P(t,n) = P(t,n-1)$, reflecting the loss of a parking space due to the arrival. Figure 5a shows an example of the effect of an arrival, including an increase in the probability that the lot has no spots.

Departures produce a straightforward change to the probability distribution. When a user departs, we know at that moment that there is a free spot in the lot, so we can set $P(t,0) = 0$ and renormalize the distribution. Note that, since the probability that the lot is free is $P_{free} = \sum_{n>0} P(t,n)$, at the exact time of each departure the probability that a spot is free is equal to 1. Figure 5b shows the hole in the distribution at zero created by a departure event.

Unsuccessful implicit searches, in contrast, represent weaker negative information, both because they were not observed by PocketParker and so may not have actually taken place, or because they may not have been thorough. What we want is to increase the probability that the lot is full while reflecting our current estimate of the lot. We do this by shifting the availability distribution towards full by some amount $s$, which we refer to as the *search shift parameter*. So, after an implicit unsuccessful search, we set $P(t,n) = P(t,n-s)$, with $P(t,0) = \sum_0^s P(t,n)$. Figure 5c shows how the distribution shifts towards zero and probability accumulates in the full state after an unsuccessful search. The search shift parameter determines how aggressively PocketParker will use information provided by implicit searches.

### 4.7.1 Weighted arrivals and departures

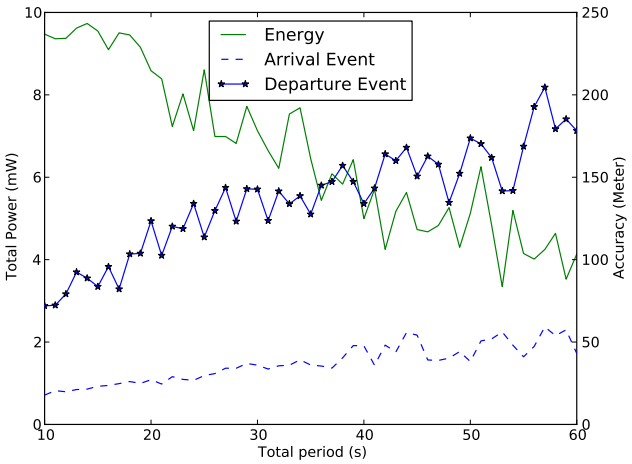



Figure 7: **Power usage vs. detector accuracy.** Energy usage by PocketParker is low at all duty cycles, so we chose a high duty cycle in order to improve detection accuracy.

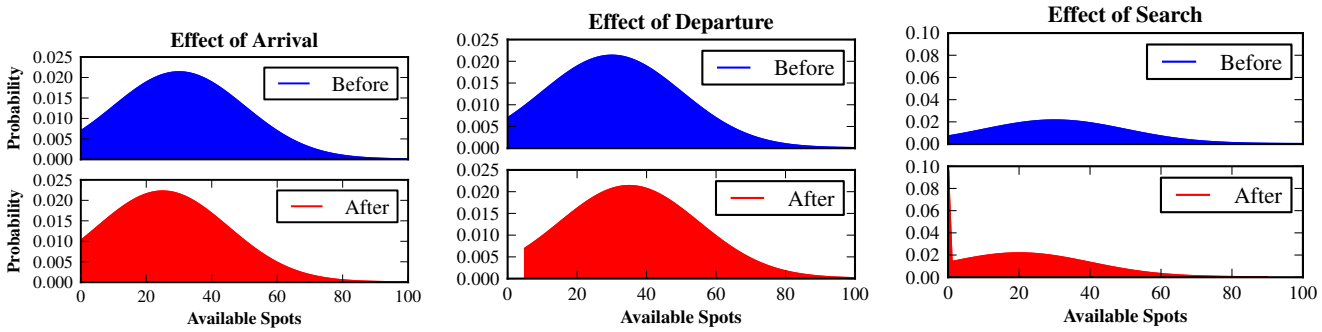
Shifting the distribution one space on arrivals and departures is the most conservative approach representing what we definitely know: that one spot is available. However, if we assume that our monitored drivers are representative of some larger number of hidden drivers, we may set $P_l(t, n < X) = 0$ for some $X$ larger than 1 and scaling with $\frac{1}{f_m}$. For our experiments we choose the conservative approach and set $X = 1$. We discuss in Section 6 how users may customize the behavior of PocketParker to be more or less aggressive in locating parking spots, trading off time for a better spot.

# 5. EVALUATION

We evaluated PocketParker in three ways. First, we conducted a controlled experiment to determine the best parameter settings for our event detector. Second, we implemented a parking lot simulator to experiment with various kinds of lots under differing monitored fractions. Finally, we performed a large-scale deployment of PocketParker on our campus and used it to monitor two lots. Camera monitoring was used to ground truth the predictions from our deployment dataset. Our evaluations confirm that PocketParker is efficient and accurate.

## 5.1 Detector Experiment

To determine the right parameter settings for our transition detector, we conducted a controlled experiment. During this experiment, accelerometer and GPS data was collected and stored continuously on each device, and participants were asked to manually label each transition into and out of the car. Afterwards, data was processed by a Python simulator implementing the identical algorithm used by the PocketParker applica-




(a) Effect of arrival. (b) Effect of departure. (c) Effect of search.

Figure 5: **Effect of different types of events on the lot availability distribution.** Arrivals, departures, and implicit searches each have a different instantaneous effect on PocketParker's availability distribution.

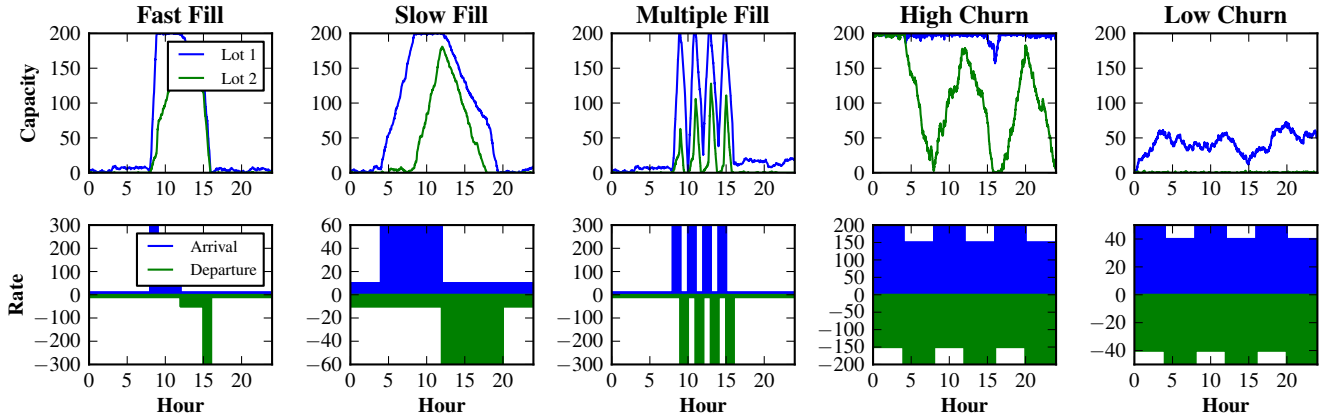



Figure 6: **Description of each type of lot simulated.** Five different lots with different behaviors were used during simulations.

tion, allowing us measure accuracy and energy consumption as a function of the detector duty cycle.

Eight volunteers participated, including seven men and one woman. Seven were right-handed and one was left-handed. Each was asked to conduct the same experiment ten times: (1) carrying the instrumented phone, walk to their car; (2) label departure; (3) drive around campus briefly; (4) park and label arrival; (5) return inside. Since the way the phone is carried while walking and placed in the car while driving affects the accelerometer readings, care was taken to generate a good mix of carry and car location styles. Table 1 shows the breakdown. The experiment permitted us to obtain sensing data from a cross section of individuals possessing different body morphologies, habits of driving cars, and ways of handling mobile devices.

Figure 7 displays the tradeoff between energy usage and detection accuracy as a function of the PocketParker duty cycle. Here we combine an active period of 5s with a inactive period of variable length, between 5 and 55s, for an overall duty cycle between 0.5 and 0.06. Our simulator uses energy numbers from the Android Fuel Gauge application o estimate average power consumption. This graph measures the accuracy of detected events in terms of distance from the actual location of the event labeled by the participant.

As we expect, longer duty cycles consume less energy but produce longer detection latencies which translate into higher distances from the event location. Note also that the departure events have higher location error than the parking events, because departing users are driving and therefore traveling more rapidly. Overall power usage by PocketParker is low, under 10 mW at all duty cycles. Because PocketParker's ability to map parking events into lots is affected by the detection distance accuracy, we chose a low total period of 15 s for a 0.25 duty cycle. This allows PocketParker to determine location to within 25 m for arrival events and 80 m for departures. Power consumption at this duty cycle is 8 mW, representing 4.2% of the capacity of a 1500 mAh battery over 24 hours of usage.

Using the same data we also examine the false posi-

| Carry Location | Count |
|---|---|
| In hand | 18 |
| Side bag | 10 |
| Back pack | 10 |
| In hand talking | 7 |
| Front pocket | 14 |
| Jacket pocket | 14 |
| Back pocket | 7 |
| **Car Location** | **Count** |
| Cup holder | 16 |
| Car seat | 9 |
| Side bag | 10 |
| Back pack | 9 |
| Front pocket | 14 |
| Jacket pocket | 14 |
| Back pocket | 8 |

Table 1: **Carry and Car Location for Controlled Detector Experiment.** Eight participants generated 80 runs, carrying the phone and placing the phone in their car in many ways.

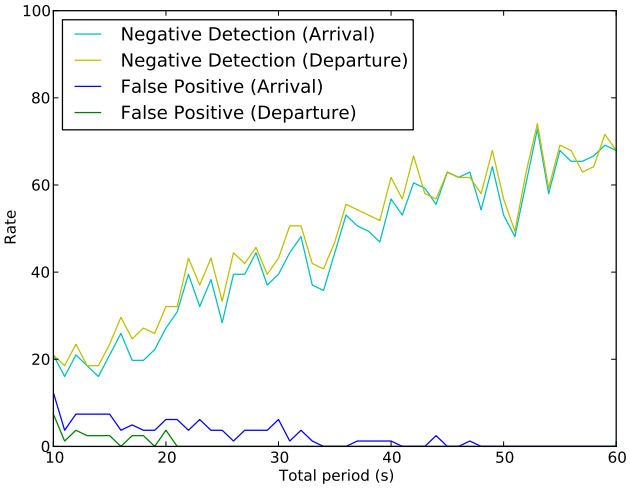



Figure 8: **False positive and negative rates as a function of detector duty cycle.**

tive and negative rates for arrivals and departures. This is important since, without explicit user input, it would be impossible to determine this information while PocketParker is in use. Figure 8 shows PocketParker can detect 80% of arrival and departure events correctly at the 0.25 duty cycle we use. False positive rates are already quite low, and this is before we apply our GPS availability filter and lot location filters. False positives decline as the duty cycle decreases because PocketParker has fewer opportunities to detect user activity.

Figure 9 shows that PocketParker detects parking events faithfully: 80% of users missed less than 25% parking events. The main reason is because PocketParker has transition time threshold of 5 minutes. Therefore, PocketParker requires a minimum of 5 minutes to detect an user transition event (from walking to driving

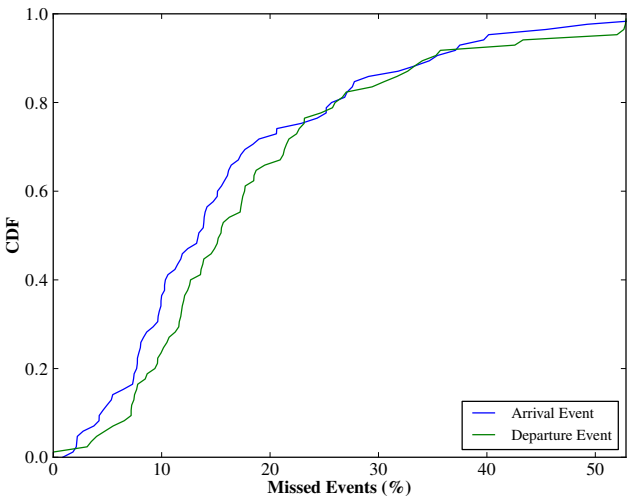



Figure 9: **The percentage of missed parking Events.**

or driving to walking).

## 5.2 Simulation Results

To experiment with PocketParker in a more controlled setting, we implemented a parking lot simulator in Python. Our simulator allows us to simulate any number of parking lots associated with any number of points of interest with varying desirability levels. For simplicity during our evaluation, we simulate two lots 1 and 2 with lot 1 filling before lot 2, although lot choice by simulated drivers is randomly weighted. Particularly for evaluating our monitored fraction estimation, we use five types of lots that fill and empty differently:

- **Fast Fill** and **Slow Fill** fill once per day quickly or slowly, like a lot associated with a place of work.
- **Multiple Fill** represents a lot that rapidly fills and empties repeatedly during each day, like a campus lot or movie theater.
- **High Churn** starts with lot 1 full and experiences continuously high arrival and departures rates, like an airport parking lot.
- **Low Churn** represents underutilized lots that never completely fill, with lot 2 almost completely unused.

Figure 6 shows the arrival and departure rates for each of the types of lot as well as the resulting per-lot capacity.

### *5.2.1 Monitored fraction estimation*

In Section 4.5.2 we describe our approach to estimated the monitored fraction, a parameter important

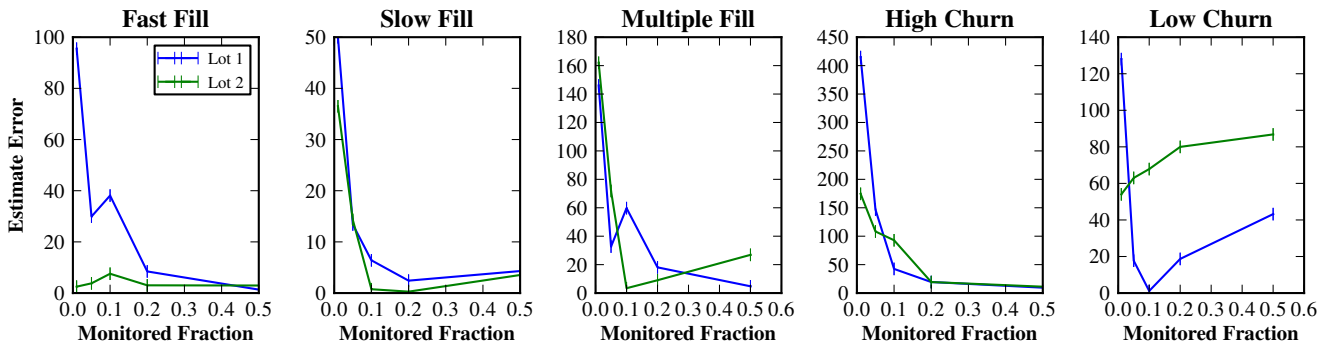



Figure 10: **Errors in monitored fraction estimation.** Currently PocketParker is better at estimating the monitored fraction when lots fill and empty regularly.

to the operation of the PocketParker availability estimator. Figure 10 shows the results of 10 random simulations for each lot type. In each case, the monitored fraction estimator uses a weeks worth of data and proceeds as described previously. The error in the monitored fraction estimate is shown as a function of the actual monitored fraction for the simulation used.

For the five types of lots, we would expect PocketParker to do better monitored fraction estimation when lots fill regularly—Fast Fill, Slow Fill, and Multiple Fill—and poorly when they do fill erratically or not at all—High and Low Churn. The results in Figure 10 generally follow this pattern. Errors for High Churn are quite high, and Low Churn errors persist even at high monitored driver fractions. This is natural, as the Low Churn lot never fills. By contrast, the accuracy rate for the Fast, Slow and Multiple Fill models improve with an increasing fraction of monitored drivers.

### 5.2.2 *Probability and availability*

At this point we take a closer look at the way that PocketParker adjusts lot availability probabilities. Keep in mind that the absolute value of the availability probability for each lot may not meaningful. Instead, PocketParker uses the probabilities to order available lots in response to queries. We examine its accuracy at performing this essential task next. However, it is illustrative to examine the probabilities PocketParker maintains and observe how they vary as the number of available spots in the lot changes.

Figure 11 shows a 24 hour simulation of a Fast Fill parking lot with a monitored fraction of 0.1 and a 10% error in the estimation of the monitored fraction. The ground truth capacity of the lot as simulated is plotted next to the PocketParker probability that the lot has an available spot. At the beginning of the simulation, both lots are marked as free. When lot 1 fills and lot 2 begins to fill, generating implicit searches in lot 1, the availability probability of lot 1 drops. It spikes upward repeatedly due to departures from lot 1—which reset

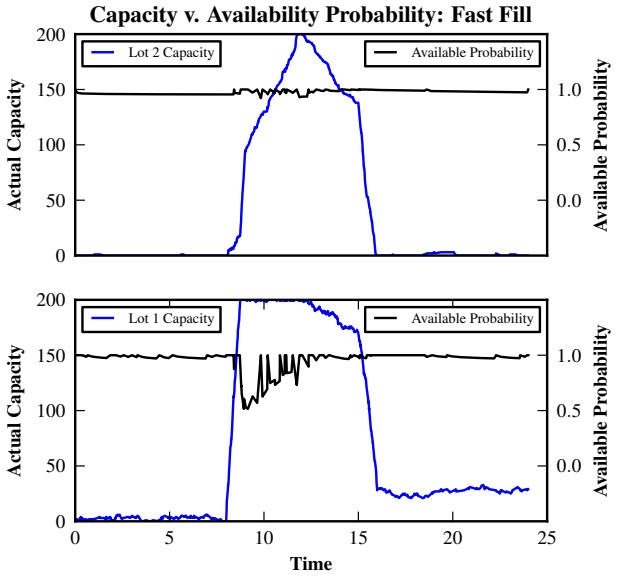



Figure 11: **Availability probabilities tracking lot capacity.** Dips in the availability probability correspond to times when PocketParker believes the lot is full. Discontinuities are caused by departures, which set the instantaneous probability that the lot is available to 1.0.

| Type | $f_m$ | Correct | Missed | Waste |
|---|---|---|---|---|
| **Campus** | 0.07 | 56.1 % | 43.9 % | 0.0 % |
| | 0.13 | 80.9 % | 1.9 % | 17.2 % |
| | 0.17 | 72.4 % | 11.0 % | 16.6 % |
| | 0.20 | 94.2 % | 5.8 % | 0.0 % |

Table 2: **Accuracy of PocketParker predictions for various fraction of monitored drivers.**

the short-term probability of an available spot back to 1—but does not equal the probability for lot 2 again until the point when the departure rate for lot 1 climbs.

### 5.2.3 *Prediction accuracy*

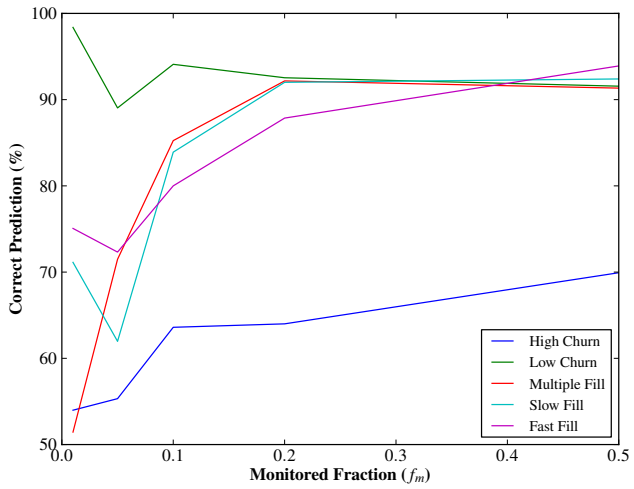



Figure 12: **Accuracy predictions for various kind of lots and parameters.**

PocketParker exists to help drivers choose parking lots efficiently. Here, we examine the accuracy of the predicitions generated by our system. To do so, we have PocketParker rank two model lots in order of preference at regular timesteps. We compare these results with the ground truth available through a simulator and then categorize them as be in a correct prediction, a missed opportunity, or a waste of time. A missed opportunity represents a case where a more desirable lot was available than the one that PocketParker recommended. A waste of time indicates that PocketParker sent the user to a lot that did not actually have an available spot. Table 2 shows data results from simulations run using varying monitored fractions $f_m$ of drivers.

Also, Figure 12 shows that several trends can be observed in the results. First, overall PocketParker does well on most lot types. The High Churn lot presents the greatest difficulty, which we would expect since its large number of incoming and outgoing drivers make prediction difficult. We are also concerned that the High Churn errors are largely waste of time errors, indicating that PocketParker is frequently sending drivers to the wrong lot. This is likely because it is predicting that spots are available longer than they actually are. Clearly more work is needed to determine the right approach for High Churn lots.

Excluding the High Churn lot, the lot with the lowest correct percentage with a $f_m > 0.1$ is 80% for the Slow Fill lot. Accuracy for all lots above this $f_m$ is consistently good for all lots save the High Churn model. The Low Churn lot does have a small number of errors but this is because both lots are usually empty.

One unavoidable lower bound to the accuracy of PocketParker is imposed by the frequency of parking events. This is because PocketParker has the most information about lot availability during active periods of arrival and departure events. Once it stops receiving event information, prediction uncertainty grows. Thus, to the degree that PocketParker queries follow at least pattern of arrivals and departures, we will have fresh data and do well.

## 5.3 Deployment

Finally, to establish the accuracy of PocketParker we performed a pair of deployments on our university campus. The system structure was the same in both cases: The only infrastructure required was the PocketParker server for receiving events and generating availability estimates. For the first rollout, we installed the PocketParker client on the Android version 4.1 system and the Samsung Nexus S 4G smartphone. In this experiment, we configured PocketParker to operate in the background and to require no user interaction. We recruited five participants who generated 372 events over ten days—202 arrivals and 170 departures.

In our second rollout, we deployed PocketParker on the Android version 4.2 system and Galaxy Nexus smartphone. PocketParker, rather than running in the background, displayed to users a campus map showing recent parking events. The userbase involved 105 total participants, 102 of whom were members of PhoneLab, an existing campus mobile phone network. Over 45 days of monitoring, they generated 10,827 events – 5916 arrivals and 4911 departures – for an average of 241 per day. Our main and medical campuses produced 3645 and 846 total events respectively, with non-campus locales contributing the remaining 6336 events.

Figure 13 shows all of the events that occurred in three key lots that we monitored in the second experiment. Our computer science building is labeled as the point of interest (POI). To determine ground truth availability, we positioned four cameras at locations within the building to monitor lots A and B in Figure 13. Despite the fact that many parking events took place in lot C, we were unable to locate a suitable unobstructed vantage point to gather camera data for that lot. Nexus S 4G smartphones equipped with fish-eye lenses served as our cameras. Each took time lapse images at 1 Hz, time-stamped them using NTP and uploaded them to a central server. A total of 34,138 images were collected for the two monitored lots over two weeks.

To measure capacity, we hand-coded four days' worth of images for two lots on a ten-point scale at ten-minute intervals. We were particularly interested in the transition between empty and full states, so we were careful to ensure that the lot was never marked completely full if there was a single available spot visible. We generate 4 different dataset using parking events in camera-monitored lots A and B and then feed the events into

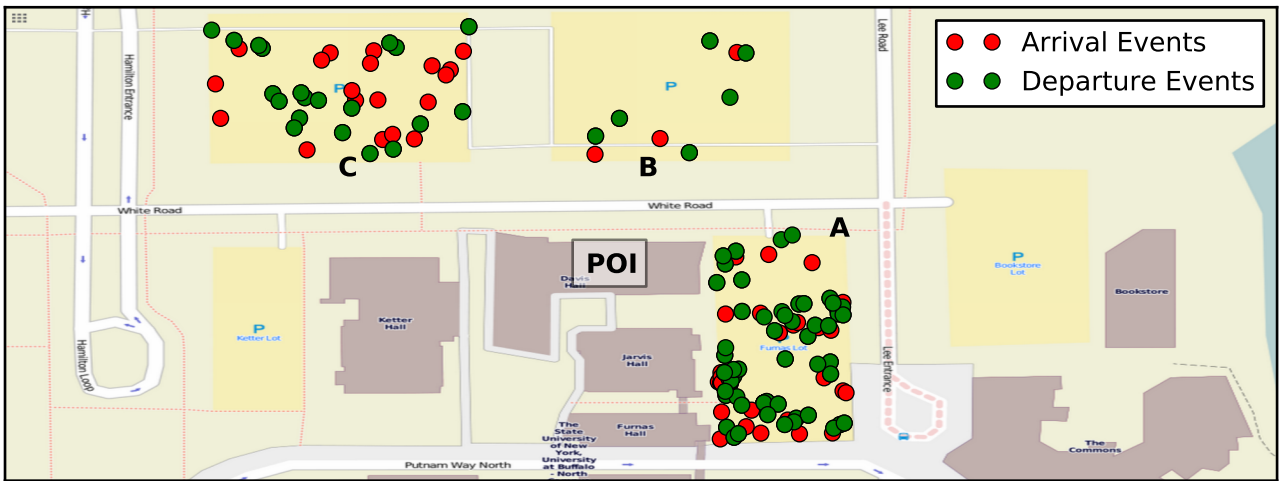



Figure 13: **Map showing 217 parking events detected by PocketParker during our forty-five-day deployment in three key lots.** These were generated by 26 participants. Lot A is considered the most desirable of the three lots, a fact reflected in the higher event density of this lot. Lots A and B were monitored by cameras to establish ground truth

the PocketParker estimation engine.

Table 2 also includes numbers for our campus deployment labeled as "Campus". Overall the accuracy of PocketParker is excellent, achieving 94.2% accuracy at a monitored driver fraction of 0.2.

## 6. LIMITATIONS AND FUTURE WORK

The pocketsourcing approach taken by PocketParker makes it easy to integrate into existing mapping applications, such as Google Maps. Doing so would benefit PocketParker in two ways. First, Google Maps and other navigation tools are in extremely wide deployment, with the Play Store estimating millions of installs for Google Maps. If we can increase the monitored fraction significantly, much of the work PocketParker does to perform estimation and work around low monitored fractions will be unnecessary.

PocketParker will also benefit from the increased amount of location context available through integration into mapping software. We imagine a "Help Me Park" button which engages PocketParker. This small piece of natural user input allows PocketParker to identify *explicit* searches and use them to build up a lot desirability model with requiring annotations. However, once PocketParker begins guiding users to available parking spaces we will have to incorporate the effects of our guidance on natural user behavior. However, we believe that many users will only query PocketParker when parking in unfamiliar locations, while still providing data about lots they use regularly and know well.

PocketParker presently bases its parking predictions on a fifteen-minute limited rolling window of recent parking events. We do not presently tap the benefit of daily and weekly patterns that would otherwise enhance predictive accuracy, but hope to do so in the future. Maintaining a database of previously collected historical data from our own application will increase the sample size and hence statistical accuracy of our parking predictions. This is another area where integration with a mapping application would help, providing PocketParker with access to much more data.

Access to historical data will also address a present fundamental limitation: the situation of a lot that fills abruptly, a typical occurrence at universities during class changes. Basing a negative recommendation about a particular lot's availability solely upon recently acquired unsuccessful searches implies that a time lag necessarily exists between when users start discovering on their own that a lot is full and when we have collected enough data to conclude—somewhat belatedly—that a lot is an unwise option. Having historical data on hand will dissolve this limitation immediately: using previous trends, we will be able to time parking advisories for particular lots before they hit capacity.

Finally, we believe that once users begin interacting with PocketParker we will see different preferences emerge. Some user will want PocketParker to help them aggressively hunt for spots, and be willing to wait for drivers to leave. Others may be more interested in simply finding a spot quickly even if it is farther away. PocketParker has several parameters that can control its predictions, and we will need to determine which are the most intuitive to users.

## 7. CONCLUSION

We have presented PocketParker, a pocketsourcing solution for predicting parking lot availability. PocketParker requires no explicit user input and can provide

parking lot predictions without being removed from a user's pocket. PocketParker's accuracy derives from combining a simple and energy-efficient parking event detector with a sophisticated parking lot availability model that incorporates the effect of hidden drivers that compete with PocketParker users for parking spots. Our evaluation has demonstrated that PocketParker can provide accurate predictions across a variety of parking lot types and patterns, and that a fielded deployment of PocketParker performed extremely well. We look forward to integrating PocketParker into existing mapping applications and bringing it to pockets everywhere.

# 8. REFERENCES


[1] CTIA: 96 million smartphones in US. http://mobihealthnews.com/13796/ctia-96-million-smartphones-in-us/, 2011.

[2] Quick, Find a Parking Space. http://goo.gl/Ybpyj, 2011.

[3] How to Find a Parking Spot on Campus. http://www.wikihow.com/Find-a-Parking-Spot-on-Campus, 2012.

[4] Parking Lot Design Standards. http://goo.gl/v0F7u, 2012.

[5] OpenStreetMap. http://www.openstreetmap.org/, 2013.

[6] Recognizing the user's current activity, Dec. 2013.

[7] G. Ananthanarayanan, M. Haridasan, I. Mohomed, D. Terry, and C. A. Thekkath. Startrack: a framework for enabling track-based applications. In Proceedings of the 7th international conference on Mobile systems, applications, and services, MobiSys '09, pages 207–220, New York, NY, USA, 2009. ACM.

[8] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson. Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11, pages 68–81, New York, NY, USA, 2011. ACM.

[9] M. Caliskan, A. Barthels, B. Scheuermann, and M. Mauve. Predicting parking lot occupancy in vehicular ad hoc networks. In Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th, pages 277–281, 2007.

[10] X. Chen, E. Santos-Neto, and M. Ripeanu. Crowdsourcing for on-street smart parking. In Proceedings of the second ACM international symposium on Design and analysis of intelligent vehicular networks and applications, DIVANet '12, pages 1–8, New York, NY, USA, 2012. ACM.

[11] M. Chester, A. Horvath, and S. Madanat. Parking infrastructure: energy, emissions, and automobile life-cycle environmental accounting. Environ. Res. Lett., 5(034001), July - September 2010.

[12] 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, MobiCom '10, pages 149–160, New York, NY, USA, 2010. ACM.

[13] T. Delot, N. Cenerario, S. Ilarri, and S. Lecomte. A cooperative reservation protocol for parking spaces in vehicular ad hoc networks. In Proceedings of the 6th International Conference on Mobile Technology, Application and Systems, Mobility '09, pages 30:1–30:8, New York, NY, USA, 2009. ACM.

[14] M. Keally, G. Zhou, G. Xing, J. Wu, and A. 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.

[15] R. Lu, X. Lin, H. Zhu, and X. Shen. Spark: A new vanet-based smart parking scheme for large parking lots. In INFOCOM 2009, IEEE, pages 1413–1421, 2009.

[16] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, and W. Trappe. Parknet: drive-by sensing of road-side parking statistics. In Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, pages 123–136, New York, NY, USA, 2010. ACM.

[17] S. Nawaz, C. Efstratiou, and C. Mascolo. Parksense: A smartphone based sensing system for on-street parking. In Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM.

[18] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using mobile phones to determine transportation modes. ACM Trans. Sen. Netw., 6(2):13:1–13:27, Mar. 2010.

[19] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys '10, pages 85–98, New York, NY, USA, 2010. ACM.

[20] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys '09, pages 85–98, New York, NY, USA, 2009. ACM.

[21] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In Proceedings of the 7th international conference on Mobile systems, applications, and services, MobiSys '09, pages 179–192, New York, NY, USA, 2009. ACM.

[22] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In Wearable Computers (ISWC), 2012 16th International Symposium on, pages 17–24, 2012.

[23] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin. Detecting driver phone use leveraging car speakers. In Proceedings of the 17th annual international conference on Mobile computing and networking, MobiCom '11, pages 97–108, New York, NY, USA, 2011. ACM.

[24] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys '12, pages 379–392, New York, NY, USA, 2012. ACM.