PocketParker: Pocketsourcing Parking Lot Availability

Paper # 166

Abstract

Searching for parking spots wastes time and generates frustration and harmful vehicle emissions. 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, and can run effectively without the phone leaving the user's pocket. Users interact with PocketParker only when looking for parking spots.

PocketParker detects parking and leaving events by monitoring the smartphone's accelerometer and using a simple and energy-efficient activity detection algorithm. Detected events are sent to the PocketParker server, which incorporates them into its parking lot availability model, allowing it to make predictions in response to availability queries. By estimating the number of *hidden drivers*—those not using PocketParker—we can use 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 can accurately and rapidly detect parking events, and that our availability estimator is accurate and robust to the presence of hidden drivers. Finally, we use camera monitoring of several parking lots and a small-scale deployment 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 [4], and the Wall Street Journal has published an online article [3] 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 [12, 13, 15, 20, 21], but include requirements rendering them impractical, such as additional infrastructure [20], on-vehicle equipment [21] or vehicular networking [15, 21], or onerous manual user input [13]. 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 [2]. PocketParker runs unattended in the background and uses the accelerometer to detect parking lot arrivals and departures, which are forwarded to a central server. There they are incorporated into per-lot availability models allowing PocketParker to accurately order lots 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 Pocket Parker—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 assist users.

SenSys'13, November 11–15, 2013, Rome, Italy.

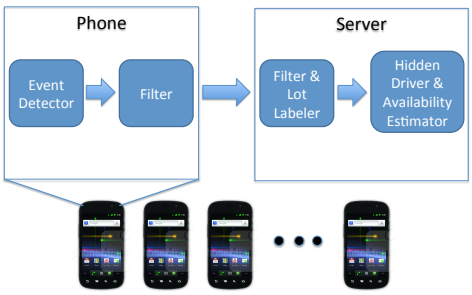


Figure 1: **The PocketParker architecture.** Events generated by an activity detector running quietly on each smartphone are processed by a central server and used to estimate parking lot availability.

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 five smartphones used by our participants over ten days. To obtain ground truth, we deploy four cameras that monitor two parking lots over two weeks. We inspect and hand-code one day's worth of images for two parking 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 the two major components of PocketParker: our parking event detector and availability model. Our evaluation in Section 5 uses several approaches to evaluate PocketParker, including controlled experiments, simulations, and a small deployment. 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 [14, 18, 23, 28, 26] have proposed techniques for activity detection, solving the main challenges of energy efficiency and accuracy. Yang et al. [28] develop an algorithm that detects if a driver is using a phone by sending high-frequency beeps via in-car speakers. EEMSS [26] is a system that provides continuous identification of general human states such as walking, driving, and being in an office. Reddy et al. [23] propose a classification approach that determines human movement states such as walking, running, biking, or vehicle-traveling. Constandache et al. [14] use smartphone accelerometers to determine users' walking trails. Yan et al. [27] propose an adaptive approach that dynamically adjusts the accelerometer sampling frequency for conserving energy. Keally et al. [18] 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 allows reservation of a parking spot. Some applications also report parking lot availability based on publicly available information. A few applications [5, 7] make use of extensive sensor deployment to monitor parking spaces on busy city roads. To the best of our knowledge, there is no application that automatically infers parking lot availability by monitoring drivers.

Perhaps closest to our work is ParkNet [21], a system that estimates street parking availability. ParkNet uses vehicles equipped with GPS and an ultrasonic range finder that scan their surrounding areas and detect empty street parking spots. Compared to ParkNet, our approach relies only on smartphones and does not require any additional input.

A few systems have been proposed to assist parking via vehicular ad-hoc networks or crowdsourcing. SPARK [20] employs extra roadside infrastructure that monitors parking lots and communicates with vehicles. Delot et al. [15] propose a parking lot reservation system in a vehicular network. Chen et al. [13] propose a crowdsourcing approach that asks participants to report the parking availability of their vicinity. Caliskan et al. [12] 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 capacity, occupancy, arrival rate, and 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, their approach assumes that each parking lot accurately measures the necessary information.

Google has operated a service called Open Spot for roughly two years but has terminated it in 2012 [1]. It was a crowdsourcing approach that relied on reports from users on free parking spots. The exact reason for this discontinuation of service is not reported.

2.3 Tracking-Related Projects

A few previous systems have investigated techniques for automatic transit tracking [11, 24, 29]. From a 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.

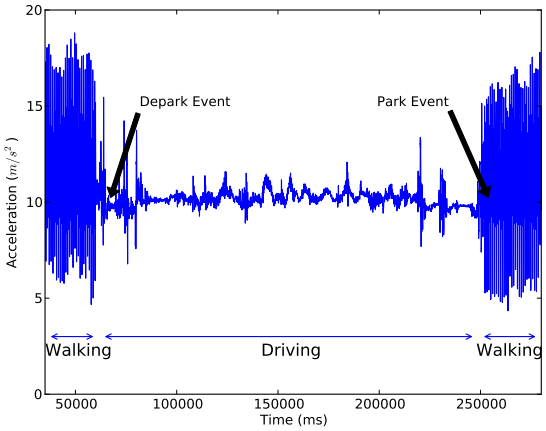


Figure 2: **Detection algorithm example.** The graph shows data collected during our controlled experiment and shows a period of walking, followed by driving, followed by a return to walking. Transitions between these states indicate parking lot arrivals and departures.

Thiagarajan et al. [24] 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 [11] uses smartphones deployed in buses to enable automatic transit tracking. The goal of EasyTracker is to require no other input than what is available from the deployed smartphones, which is similar in spirit to our goal of pocketsourcing. The system combines a few mechanisms to realize this goal such as route extraction, stop extraction, and arrival time prediction. Zhou et al. [29] 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 [25] is a traffic monitoring system that combines readings from multiple sensors for travel time estimation. Nericell [22] uses multiple sensors to monitor road and traffic conditions in an urban setting. P^2 [16] uses vibration and GPS sensors to monitor road surface conditions. Balan et al. [10] use GPS devices deployed in taxi cabs to estimate taxi fare and trip duration. StarTrack [9, 17] provides general abstractions for applications that need tracking functionalities such as recording, comparing, and querying tracks. SignalGuru [19] uses smartphones with cameras mounted in vehicles to learn traffic signal schedule patterns, so drivers can adjust their speed.

3 Event Detector

The inputs to PocketParker's availability estimation algorithm are arrival and departure events generated by an activity detector running unattended in each user's pocket. While a great deal of previous research has explored activity detection using sensors on mobile devices [14, 18, 23, 28, 26], we design a simple custom parking event detector tailored to the goals of PocketParker. In addition to being designed and tuned to the specifics of our detection problem, our event detector can also utilize the structure of our monitoring problem to reduce false positives. In this section, we describe our event detector and the portions of the PocketParker system that run on the smartphone itself.

3.1 Parking Events

PocketParker assumes that transitions between walking and driving that occur in locations known to be parking lots constitute either arrival (driving to walking) or departure (walking to driving) events. Thus, we must be able to efficiently detect walking and driving activities, and detect transitions fast enough to determine the location of the parking lot in which the event took place. While all these states could be detected using continuously-sampled GPS data, this would consume too much energy to be an effective pocketsourcing solution. Instead, we rely on duty-cycled accelerometer data to classify the user into one of three states: walking, driving, or idle.

The application needs to determine the state of the user—whether walking, driving or idle—with a test that is simple, quick and reasonably effective for our goals. As we show in Section 5, an accelerometer $\frac{1}{4}$ duty cycle with 5 s sensing and 15 s idle periods proved optimal and five seconds was the minimum time necessary to detect sensing activity accurately.

In order to set the initial state of a user, we use the standard deviation of sampled accelerometer data within a sensing period. Our application first feeds readings through a smoothing filter to suppress ambient sensor noise and other low amplitude signals. A standard deviation of the processed signal of less than 0.15 $\frac{m}{s^2}$ implies a user state of idle, a measurement above 1.0 $\frac{m}{s^2}$ to that of walking, and values in between to driving. The application stores a history of user states. From this history, we define the user steady state to be a state, if any, that constitutes a majority of the individual states observed within the last nine periods, an empirically-determined parameter proven to be effective.

We use user state, in turn, to deduce parking arrival events. We determine that a parking event has occurred based on several conditions. The user steady state must be driving and the last two individual states observed must be walking. Alternatively, if the steady state is driving but only the last one state observed is walking, we impose an additional constraint: the number of waveform peaks 2 $\frac{m}{s^2}$ above the mean observed accelerometer value must exceed a minimum threshold corresponding to that produced by a slow human gait. This test catches a large majority of parking events, where people park and immediately begin walking.

However, drivers occasionally sit in their cars for a period before leaving. To detect this type of parking event, we employ a second test. The difficulty is that simply observing a state transition from driving to idle is not, by itself, reliably determinative. While the walking and idle states exhibit consistent, well-defined patterns, the driving state presents with more fluctuation, i.e., the steady state of driving often

contains individual idle state periods. Due to this difficulty, we determine that a driver has parked only when the user's steady state changes from driving to idle to walking within a three-minute period.

When we determine that a user has parked, we turn on GPS detection long enough to determine the walking velocity and bearing of the user. We then calculate backwards to estimate the location where the user started walking—i.e., the parking spot. We then report the occurrence and location of the newly detected parking event to the server. The server verifies the reported event against a pre-compiled list of known parking lot locations before recording the event. 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).

Occasionally, a user upon parking will walk for too brief a period before entering a building for the program to obtain a location from GPS. We find that at least in our evaluation settings, this is an uncommon occurrence as we have few parking spots close enough to buildings to produce this behavior. Nevertheless, a reliable watermark for this rarity is a steady driving state coupled with a most recent individual walking state and a null GPS reading; thus, we employ alternative Android location providers such as WiFi to determine user locale.

While walking, users may change their bearing. If this occurs before the first GPS fix, the app may not be able to calculate the original location of the parking spot. An apparent remedy to this would be to use compass data to detect changes in bearing. This would work if the relative orientation of the phone viz-a-viz the user were also detectable. In practice though, when not in use mobile phones are typically oriented randomly by users, particularly when placed in pockets and backpacks. Sticking solely with GPS data has thus proven better. Going forward, maintaining a history of user pedestrian history should further minimization of bearing inaccuracies.

A final potential issue with detecting a parking event is that of handling a driving period too short to for us to detect that a user is driving in the first place. For example, this situation could occur when a user simply moved his car to a new parking spot in an adjacent row. Since it does not stem from rational user behavior, we do not handle this case.

We employ user state transitions to detect parking departure events in a manner similar to that used to detect parking arrivals. We tentatively detect a departure event when the user's steady state is walking but the two most recent individual states are driving. The constraint of two consecutive driving states rather than one proved necessary to minimize spurious detection of driving due to the relatively variable nature of accelerometer-determined driving sense data.

Before reporting a departure event, we employ GPS sensing to confirm a driving state. In theory, it is unnecessary to turn on the GPS system, as we have already previously determined the parking spot location when the user first parked. However, we have still found that doing so aids ferreting out remaining falsely detected driving states. A null GPS reading implies that a user is indoors, and that our initial inference of a driving state was misplaced. As with arrival events, departure event locations are matched against a known list of lot locations to filter false reporting.

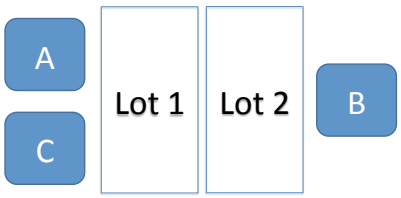


Figure 3: **Example parking lot setup.** Two lots and three destinations are shown.

3.2 Filtering False Positives

Incorrect parking arrival and departure events stem in part from the underlying issue of false detection of the individual driving state. The relatively variable accelerometer data patterns produced when driving—and that are often counterfeited by intermittent walking—complicate reliable detection of this state. Luckily, a sizable bulk of these false positives consists of the case where users eventually simply walk indoors. Mandating that a valid GPS reading accompanies a preliminary driving state determination alleviates this subset, albeit at the price of additional power consumed.

False detection of both arrival and departure events could be further minimized by using more accurate but computationally intense algorithms that implement machine learning or Fourier transforms. 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.

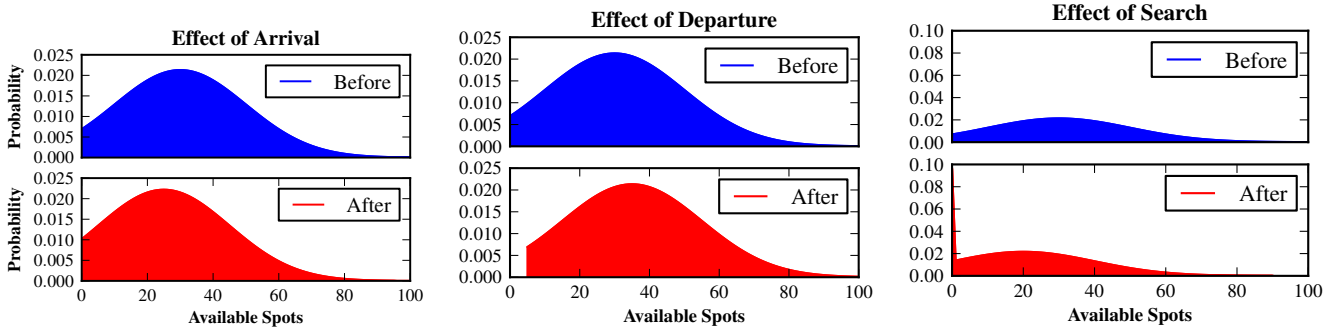
4 Availability Estimation

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 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 3 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



(a) Effect of arrival.

(b) Effect of departure.

(c) Effect of search.

Figure 4: **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.

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, detailed in Section 4.5. Recall from Section 3.2 that our false-positive filter uses knowledge of the location of lots obtained from the OpenStreetMap database [8]. We estimate the capacity of each lot by converting the location of the lot into a size and dividing by the size of an average parking spot, using a parking spot size typical to standard parking lot designs [6]. Comparing this estimate with manually-counted capacities for the five lots monitored by our deployment, capacity estimates were within 6% of the true capacity in all cases.

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 3, 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 3, 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 3 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 3, 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

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.

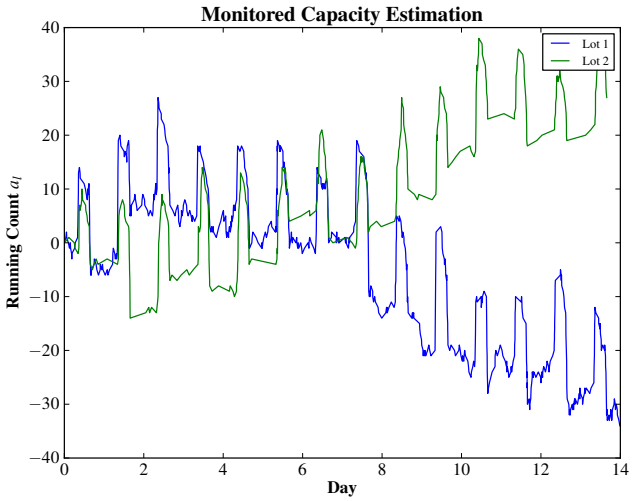


Figure 5: **Example of capacity estimation.** Running counts for two lots are shown.

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

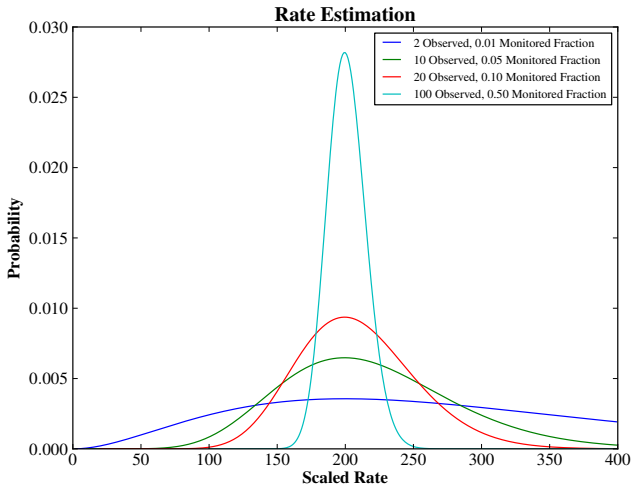


Figure 6: **Example of rate estimation.** Spread of each distribution shows the effect of the monitored fraction on rate certainty.

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 5 shows an example of the running count for 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 5, 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 similar 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 6 shows the resulting distribution for three different values of f_m with $s_{true} = 200$. While for all four values of f_m,

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

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 Δ 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 Δ 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, Δ 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 spreading 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 4 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 4a 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 4b 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 4c 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

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.

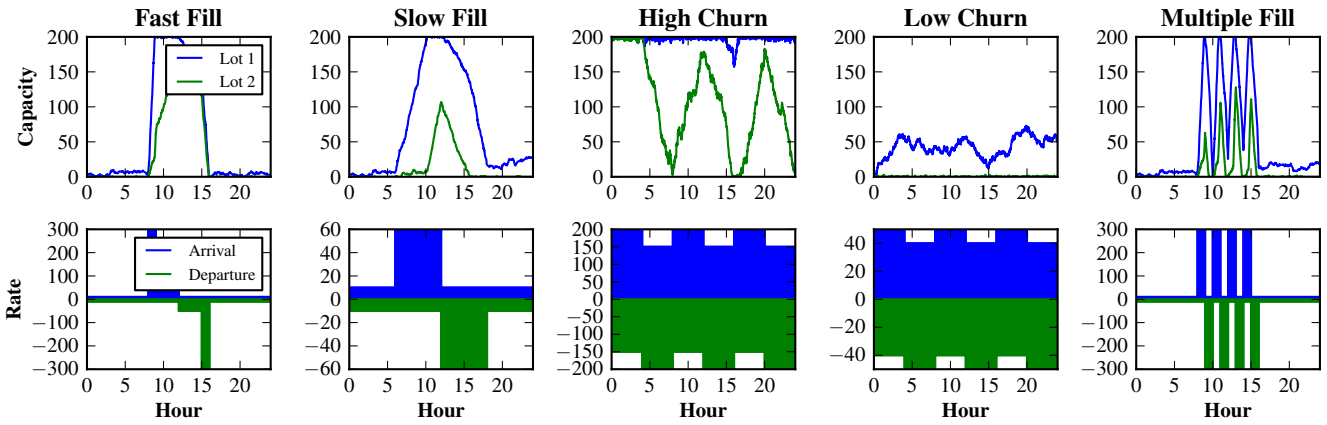


Figure 7: **Description of each type of lot simulated.** Five different lots with different behaviors were used during simulations.

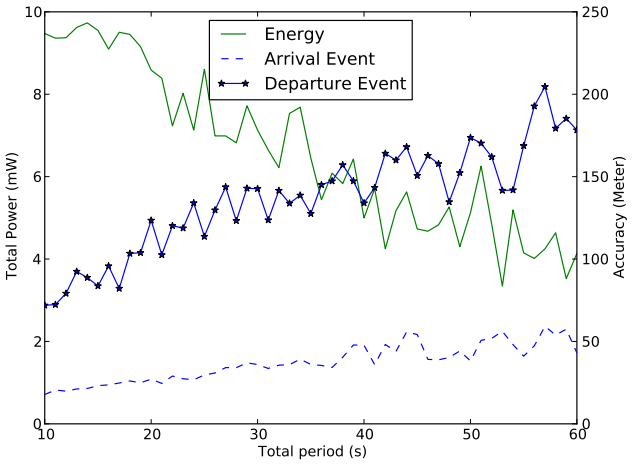


Figure 8: **Power usage vs. detector accuracy.** Energy usage by PocketParker is low at all duty cycles, so we chose a high duty cycle in order to improve detection accuracy.

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.

5 Evaluation

We evaluate 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 and vary the monitored fraction and other parameters. Finally, we performed a small-scale deployment of PocketParker on our campus and use 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 application, 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. Participation in the experiment took roughly one hour and pizza was provided.

Figure 8 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

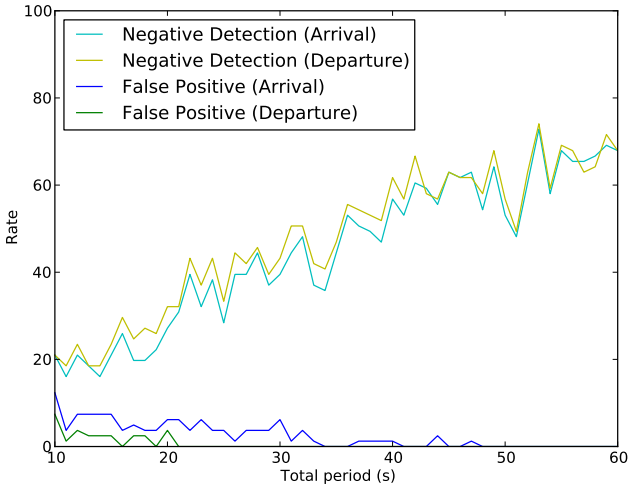


Figure 9: **False positive and negative rates as a function of detector duty cycle.**

period of variable length, between 5 and 55s, for an overall duty cycle between 0.5 and 0.06. Our simulator uses energy numbers used by the Android Fuel Gauge application along with the accelerometer duty cycles and number of GPS locations requested including false positives to 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 parking 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 positive 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 9 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.

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.
- **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.
- **Multiple Fill** represents a lot that rapidly fills and empties repeatedly during each day, like a campus lot or movie theater.

Figure 7 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 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. Fast Fill results improve with increasing monitored driver fraction. We are investigating, however, the problems with estimating lot 1 capacity, and persistent errors with the Slow Fill lots. Improving the accuracy of monitored fraction estimation may require additional smoothing techniques or merging information from multiple related lots. However, as we demonstrate later, our availability estimator is robust to monitored fraction estimation errors up to 50%.

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

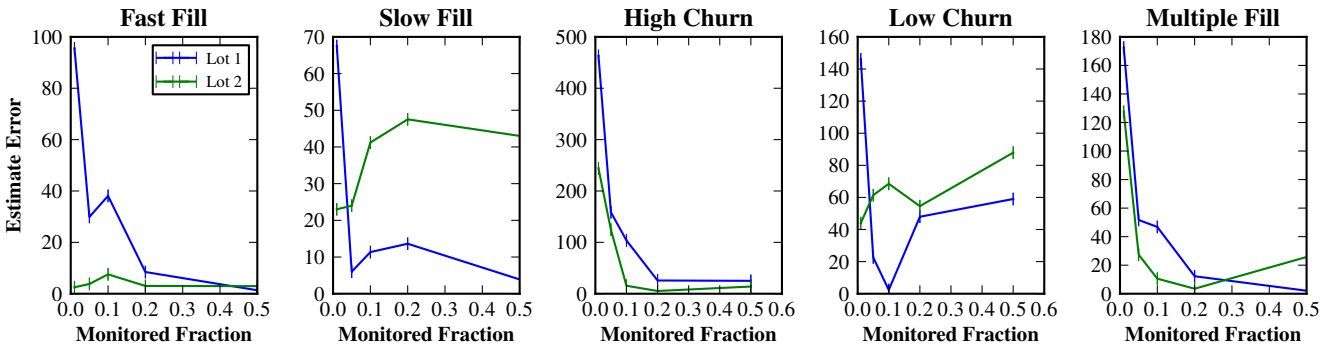


Figure 10: **Errors in monitored fraction estimation.** Currently PocketParker is better at estimating the monitored fraction when lots fill and empty regularly.

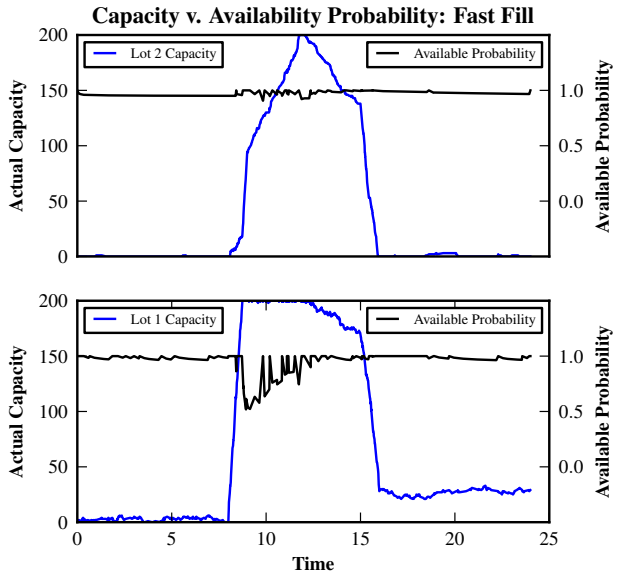


Figure 11: **Availability probabilities tracking lot capacity.** Dips in the availability probability correspond to times when PocketParker believes the lot is full. Discontinuities are caused by departures, which set the instantaneous probability that the lot is available to 1.0.

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 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

PocketParker exists to help drivers choose parking lots. Here, we examine our simulation results and determine whether we accomplish this goal. In our simulations, we inject regular queries to PocketParker asking it to order the two available lots. We categorize its response as either 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 from simulations run on several different lot types and while varying simulation parameters including the monitored fraction f_m and estimation error in the monitored fraction. At regular timesteps we query PocketParker and ask it to order the two lots lot 1 and lot 2. We then compare its ordering with the ground truth available through the simulator. In the case that two lots have probabilities within 0.01 of each other, PocketParker considers this a tie and suggests the more preferable lot first.

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 exclusively waste, indicating that PocketParker is consistently sending drivers to the wrong lot, probably 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 lowest correct percentage with a $f_m > 0.1$ is 75% for the Slow Fill lot, and accuracy is good throughout. The Low Churn lot has a small number of errors but this is because both lots are usually empty. Its error breakdown is similar to the High Churn lot, which suggests it is something about regular departures and arrivals that is causing PocketParker to be too optimistic about availability in the more desirable lot.

Note, however, that this uniform query pattern is the most challenging for PocketParker and so these results may be seen as a lower bound. This is because PocketParker has the most information about lot availability during active periods of arrival and departure events. Once it stops receiving information, uncertainty grows. So to the degree that PocketParker queries follow at least to some degree the pattern of arrivals and departures, we will have fresh data and do well.

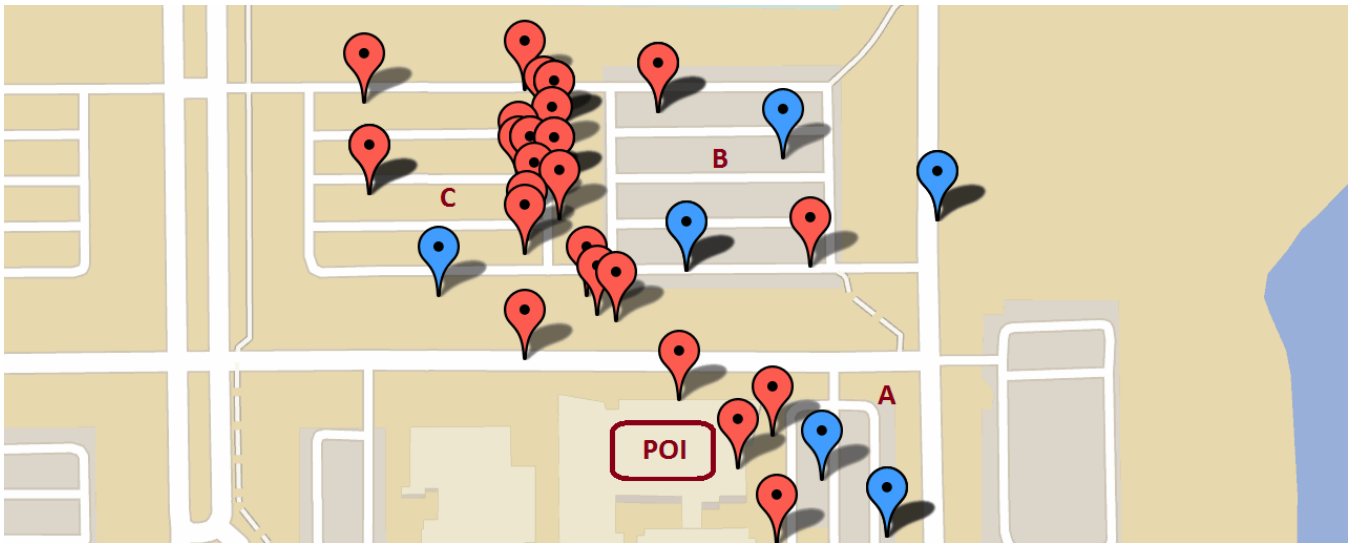


Figure 12: **Map showing all events detected by PocketParker during our ten-day deployment.** 56 events are shown in three monitored lots. Red markers indicate arrivals, blue departures. Lots A and B were monitored by cameras to establish ground truth, and Lot A is considered the most desirable of the three lots.

5.3 Deployment

Finally, to establish the accuracy of PocketParker we performed a small-scale deployment on our university campus. We implemented a PocketParker client as an Android application for Android version 4.1 "JellyBean". We tested our app on Samsung Nexus S 4G smartphones, but PocketParker should be able to run on any smartphone with GPS and an accelerometer, and on previous versions of Android. Other than the PocketParker server which receives events and updates the availability estimate, no infrastructure is required.

We recruited five participants that installed our PocketParker application and generated 372 events over ten days—202 arrivals and 170 departures. Of those events, 127 occurred near campus including false positives, and 56 were mapped into the three campus lots monitored by PocketParker[2]. The deployment period included several days when participants did not come to campus, and most parking events were generated by three participants who came to campus daily. No participants complained of excessive energy usage or battery life degradation, and no interaction with the application was required.

Figure 12 shows all of the events that occurred in the three lots we monitored. Our computer science department is labeled as the point of interest. To determine the ground truth availability of the monitored lots we positioned four cameras at locations within the building to monitor lots A and B in Figure 12. 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. We used Nexus S 4G smartphones equipped with fish-eye lenses as cameras. Each took time lapse images at 1 Hz, time-stamped them using NTP and uploaded them to a central server. Figure 13 shows two of our cameras and a view of one of the monitored parking lots. A total of 36,000 images were collected for the two monitored lots over two weeks.

To measure capacity, one day's worth of images for two lots was hand-coded on a ten-point scale at ten-minute intervals. Because we are particularly interested in the transition between empty and full states, we were careful to ensure that the lot was never marked completely full if there was a single available spot visible. Because we collected a relatively-small number of parking events into camera-monitored lots A and B during our deployment, we merge all events into a single day before feeding the events into the PocketParker estimation engine. Given the location of other campus buildings to the south of our building, lot A is considered more desirable than lot B.

Table 2 also includes numbers for our campus deployment labeled as "Campus". Because we do not know the monitored fraction, we experiment with several different values. Overall the accuracy of PocketParker is excellent, even achieving 100% accuracy at a monitored driver fraction of 0.2. We believe this is because the more preferable monitored lot has a very predictable availability pattern, filling rapidly and staying full during business hours.

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 100 500 million 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.

[2]One participant lives quite close to campus, and many of the parking transitions that occurred at their house were included in the 71 events near campus that did not map into known parking lots.

Type	f_m	f_m Error	Correct	Missed	Waste
Campus					
	0.05	0.00	81.6 %	18.4 %	0.0 %
	0.10	0.00	95.8 %	4.2 %	0.0 %
	0.20	0.00	100.0 %	0.0 %	0.0 %
Fast Fill					
	0.01	0.10	75.1 %	24.7 %	0.2 %
	0.05	0.10	72.3 %	26.2 %	1.5 %
	0.10	0.10	80.0 %	15.6 %	4.4 %
	0.10	0.20	86.6 %	11.2 %	2.2 %
	0.10	0.50	89.7 %	7.7 %	2.7 %
	0.10	1.00	93.0 %	4.8 %	2.2 %
	0.20	0.10	87.9 %	8.5 %	3.6 %
	0.50	0.10	94.0 %	5.0 %	1.0 %
High Churn					
	0.01	0.10	54.0 %	0.0 %	46.0 %
	0.05	0.10	55.3 %	0.0 %	44.7 %
	0.10	0.10	63.6 %	0.0 %	36.4 %
	0.10	0.20	62.2 %	0.0 %	37.8 %
	0.10	0.50	60.1 %	0.0 %	39.9 %
	0.10	1.00	61.8 %	0.0 %	38.2 %
	0.20	0.10	64.0 %	0.0 %	36.0 %
	0.50	0.10	69.8 %	0.0 %	30.2 %
Low Churn					
	0.01	0.10	98.4 %	0.0 %	1.6 %
	0.05	0.10	89.1 %	0.0 %	10.9 %
	0.10	0.10	94.1 %	0.0 %	5.9 %
	0.10	0.20	91.0 %	0.0 %	9.0 %
	0.10	0.50	87.9 %	0.0 %	12.1 %
	0.10	1.00	87.8 %	0.0 %	12.2 %
	0.20	0.10	92.5 %	0.0 %	7.5 %
	0.50	0.10	91.6 %	0.0 %	8.4 %
Multiple Fill					
	0.01	0.10	51.4 %	42.8 %	5.8 %
	0.05	0.10	71.2 %	26.0 %	2.9 %
	0.10	0.10	85.2 %	13.2 %	1.6 %
	0.10	0.20	79.4 %	17.5 %	3.1 %
	0.10	0.50	86.2 %	12.7 %	1.1 %
	0.10	1.00	77.6 %	20.9 %	1.5 %
	0.20	0.10	92.1 %	7.5 %	0.5 %
	0.50	0.10	91.6 %	7.8 %	0.6 %
Slow Fill					
	0.01	0.10	70.7 %	26.0 %	3.2 %
	0.05	0.10	78.8 %	17.5 %	3.7 %
	0.10	0.10	82.1 %	13.3 %	4.6 %
	0.10	0.20	75.4 %	15.2 %	9.4 %
	0.10	0.50	80.5 %	19.3 %	0.2 %
	0.10	1.00	85.2 %	12.5 %	2.3 %
	0.20	0.10	87.8 %	9.8 %	2.4 %
	0.50	0.10	93.8 %	5.6 %	0.6 %

Table 2: **Accuracy of PocketParker predictions for various kinds of lots and parameters.**

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.

Figure 13: **Monitoring cameras.** A view of one of the monitored parking lots is shown.

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.

References

[1] Open spot deprecated. http://goo.gl/aLTzd.

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

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

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

[5] Parker. http://goo.gl/yKoe2, 2012.

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

[7] SFPark. http://goo.gl/ZlxQu, 2012.

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

[9] 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.

[10] R. K. Balan, K. X. Nguyen, and L. Jiang. Real-time trip information service for a large taxi fleet. In Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys '11, pages 99–112, New York, NY, USA, 2011. ACM.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In Proceedings of the 6th international conference on Mobile systems, applications, and services, MobiSys '08, pages 29–39, New York, NY, USA, 2008. ACM.

[17] M. Haridasan, I. Mohomed, D. Terry, C. A. Thekkath, and L. Zhang. Startrack next generation: a scalable infrastructure for track-based applications. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[18] 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.

[19] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi. Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. In Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys '11, pages 127–140, New York, NY, USA, 2011. ACM.

[20] 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.

[21] 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.

[22] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In Proceedings of the 6th ACM conference on Embedded network sensor systems, SenSys '08, pages 323–336, New York, NY, USA, 2008. ACM.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.