# BreadCrumbs: Forecasting Mobile Connectivity

Anthony J. Nicholson
Department of Electrical Engineering and Computer Science
University of Michigan
tonynich@umich.edu

Brian D. Noble
Department of Electrical Engineering and Computer Science
University of Michigan
bnoble@umich.edu

## ABSTRACT

Mobile devices cannot rely on a single managed network, but must exploit a wide variety of connectivity options as they travel. We argue that such systems must consider the *derivative* of connectivity—the changes inherent in movement between separately managed networks, with widely varying capabilities. With predictive knowledge of such changes, devices can more intelligently schedule network usage.

To exploit the derivative of connectivity, we observe that people are creatures of habit; they take similar paths every day. Our system, BreadCrumbs, tracks the movement of the device's owner, and customizes a predictive mobility model for that specific user. Combined with past observations of wireless network capabilities, BreadCrumbs generates *connectivity forecasts*. We have built a BreadCrumbs prototype, and demonstrated its potential with several weeks of real-world usage. Our results show that these forecasts are sufficiently accurate, even with as little as one week of training, to provide improved performance with reduced power consumption for several applications.

## Categories and Subject Descriptors

C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

## General Terms

Management, Measurement, Human Factors

## Keywords

Connectivity forecast, opportunistic connectivity, derivative of connectivity, BreadCrumbs


*MobiCom'08,* September 14–19, 2008, San Francisco, California, USA.


## 1. INTRODUCTION

Operating systems manage wireless networks *in the moment*, reactively choosing connections only when circumstances change. This is a reasonable position to take if most users are merely nomadic, and the few truly mobile users rely on homogeneous access points.

Unfortunately, this static and simple world is fast becoming the exception, not the rule. Users demand continuous functionality while navigating a sea of diverse connection alternatives. In this environment, applications cannot make reliable assumptions about the quality of connectivity. Instead, it fluctuates based on both the path taken through uncoordinated public deployments and the varied quality of individual access points.

This setting presents both new challenges as well as opportunities. Reactive management performs poorly. Instead, one must consider the *derivative* of connectivity—how it changes over time—to properly support mobile, networked applications.

This paper describes BreadCrumbs, our system that lets a mobile device exploit this derivative of connectivity as its owner moves around the world. BreadCrumbs maintains a personalized mobility model on the user's device, and a history of observed networking conditions. Together, these predict near-term connectivity given a user's current movement. Because people are creatures of habit, these *connectivity forecasts* can be accurate with even minimal training time. Applications, or the operating system itself, can use these forecasts to defer less time-sensitive or low-priority work to a time that will improve performance, or reduce power consumption, or both.

To demonstrate the efficacy of our approach, we used a BreadCrumbs prototype for several weeks of day-to-day activity. During this time, both the quality and the availability of publicly-accessible APs were quite uneven. In spite of this, BreadCrumbs was able to predict the device's next-step downstream bandwidth from the Internet within 10 KB/s for over half of the time, and within 50 KB/s for over 80% of the time. These results were achieved with only one week of training.

We further explored how BreadCrumbs' connectivity forecasts could aid three example applications: (1) updating a handheld map application as the user moves, (2) streaming media content from a remote server, and (3) opportunistic writeback of created media content. Compared to prediction-ignorant baselines, BreadCrumbs' forecasts let all three applications improve the user experience in domain-specific ways.

This work makes the following contributions:

- We introduce the concept of *connectivity forecasts* for mobile devices.
- We demonstrate that such forecasts can be accurate over regular, day-to-day use, without requiring GPS hardware or extensive centralized infrastructure.
- We illustrate the potential benefits of the system through three example applications.

# 2. BACKGROUND

## 2.1 Determining AP Quality

There is little point to developing a complex system for forecasting the quality and availability of public wireless connections if they are few and far between, or all access points have equivalent connection quality. We explored the current state of affairs in our prior work [21], which described Virgil—an AP selection tool that considers the application-visible quality of access points. In contemporary operating systems, wireless connection managers typically select the unencrypted AP with the strongest received signal strength. Rather than consider such link-layer criteria, Virgil quickly associates with each candidate AP and runs a battery of tests designed to estimate the connection quality applications would enjoy if the device were to choose this access point.

Virgil connects to *reference servers* in order to estimate this connection quality. A reference server is a well-known Internet destination that runs a simple TCP server process. Like a honeypot, this process listens on a wide range of TCP port numbers. To probe the application-visible quality of an access point, Virgil connects to a reference server via the AP and runs the following tests:

- Estimate downstream bandwidth by connecting to the TCP server process on a well-known port and downloading random data as fast as possible.
- Determine if the AP is blocking certain services by attempting a TCP connection to common port numbers.
- Estimate latency by pinging the reference server.

We compared the success of selecting APs based on signal strength with selecting the AP with the best downstream bandwidth probed by Virgil, in five neighborhoods of varied density in three different cities in the United States. Use of Virgil resulted in a 22–100% increase in the percentage of scans that successfully found an access point with a usable Internet connection.

Much in the same way, BreadCrumbs uses a reference server to estimate the connection quality of the access points encountered by mobile devices. In addition to downstream bandwidth, BreadCrumbs also estimates upstream bandwidth via the AP. Rather than simply pinging the reference server, we estimate latency by opening a TCP connection and ping-ponging a integer nonce back and forth. This was an attempt to more closely mimic how real applications would utilize a network connection. Finally, BreadCrumbs omits the port status tests in order shorten the testing process. In summary, BreadCrumbs uses the techniques described above to estimate the following three values for each open access point the mobile device encounters: (1) downstream TCP bandwidth from an Internet host, (2) upstream TCP bandwidth to the Internet, and, (3) latency from the device to remote destinations.

## 2.2 Estimating Client Location

In order for a device to predict its future mobility, it needs some way to determine its location. This location could be descriptive ("at the Union"), relative to known locations, or absolute. In our case, BreadCrumbs uses latitude and longitude coordinates as the basic building blocks of each device's mobility model. Typically, this can be provided by GPS. Even for devices without GPS technology, it is possible to estimate one's position with reasonable accuracy, using technologies like Place Lab [16]. This project exploits the fact that a plethora of fixed-position beacons exist in the everyday environment—namely, WiFi access points and GSM mobile phone towers. A nice benefit of Place Lab is that it works well when GPS does not—indoors and in urban canyons.

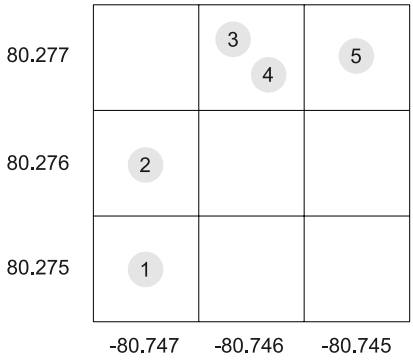
Place Lab relies on public *wardriving* databases, which map beacon MAC addresses to GPS locations. For example, `wigle.net` currently tracks over 11 million distinct access points in its database. Place Lab generates a GPS fix by first scanning for all beacons in the device's vicinity, then triangulating based on the GPS location of each beacon source. Their evaluation results (in 2005) found the mean accuracy of Place Lab's location estimates to be on the order of 20-30 meters from the GPS "ground truth" when only WiFi beacon sources were utilized. As we shall see, such error is acceptable for our needs.

# 3. CONNECTIVITY FORECASTING

By leveraging Virgil and either Place Lab or GPS data, one can determine both the locations a user has previously visited and the application-level quality of network connectivity at those locations. Our goal is to combine these two sets of data to yield what we will call *connectivity forecasts*. A connectivity forecast is an estimate of the quality of a given facet of network connectivity at some future time. An example would be the estimated upstream bandwidth from the client to a remote host 20 seconds in the future. This is a function both of the user's mobility—which APs will be in range at that time—and of the quality of these APs' network connections.

A wide variety of applications can exploit such forecasts. For example, consider a distributed file system client that needs to re-integrate some data to a remote file server. If energy consumption is a first-class concern—as it is for handheld devices—the best policy for the client would be to transmit data to the file server when the mobile device has the highest-bandwidth network connection that it will enjoy in the near future.

This section first discusses how BreadCrumbs maintains a personalized device mobility model, based on the past sequence of GPS locations the user visits. Next, we describe how BreadCrumbs applies the principles of Virgil [21] to estimate the quality of different access points, and combines this data with the predictions of the mobility model. The section concludes with a concrete example of how connectivity forecasts are generated.




| State | Last GPS | Current GPS |
|---|---|---|
| 1 | — | (80.275,-80.747) |
| 2 | (80.275,-80.747) | (80.276,-80.747) |
| 3 | (80.276,-80.747) | (80.277,-80.746) |
| 4 | (80.277,-80.746) | (80.277,-80.746) |
| 5 | (80.277,-80.746) | (80.277,-80.745) |

**Figure 1: Generating states from mobility history. Each state in the second-order Markov model encodes the current GPS location and the previous location. GPS fixes are estimated at a set period $\tau$ that is the time interval between state transitions in the model.**

## 3.1 Predicting Future Mobility

Mobility prediction is a well-studied area, particularly in the domain of mobile phone networks. The majority of applications of such techniques focus on allowing a central authority to track the movement of devices to pre-provision network resources [1, 2, 4, 17, 22, 25, 29]. As did Place Lab, we note that tracking mobility history at a central point is problematic. When such databases are compromised—either accidentally, maliciously, or under subpoena—the precise movements of users are disclosed without consent. Furthermore, mobile devices may need this information the most at precisely the times when they are disconnected from the network and cannot query the centralized server.

Synthetic mobility models [27] or aggregate models derived from the movements of many users [14, 28] are useful when a network provider needs the big picture of how their network will be utilized. However, such models have little chance of accurately capturing the very unique paths one user takes through their environment.

The most compelling reason to maintain the model on the device itself is that, unlike for a mobile phone network, there exists no one centralized authority who controls all public WiFi APs that the user encounters. This limits our choice of mobility models to those that can reasonably be maintained on resource-constrained, handheld devices. Song et al [26] previously evaluated the accuracy of several common mobility prediction models, using mobility data collected on the campus of Dartmouth College during the 2003-2004 academic year [15]. This dataset tracks the AP association history of over 7000 users to over 550 WiFi access points of known location.

Their evaluation found a second-order Markov model, with fallback to a first-order model when the second-order model has no prediction, was the most accurate of all techniques examined. Conveniently, Markov models are ideal for use on resource constrained devices. Their CPU needs are low because model querying and maintenance involves merely reading and writing individual entries in arrays. Since these arrays are generally sparse, storage requirements are modest.

We chose geographic longitude and latitude coordinates as the fundamental building block of our model. Since we have chosen a second-order Markov model, each state consists of two sets of coordinates: the location where the device was during the last state, and its current location. Tracking this second-order state is useful for distinguishing between different mobility paths that share a common point. For example, this can disambiguate between the user walking eastbound and westbound on the same street.

The resolution of our model is bounded both by the accuracy of location sensing and the resource constraints of mobile devices. To avoid a state space explosion, BreadCrumbs rounds all GPS values to three decimal places—one one-thousandth of a degree. While the size of one degree of latitude is constant everywhere on the Earth, the distance between two degrees of longitude shrinks as one moves further away from the equator. At our latitude, a $0.001° \times 0.001°$ grid square is 110 m×80 m.

While a higher degree of location precision than 110×80 meters would seem desirable, this was impractical for two reasons. First, location estimates inherently have some amount of error from the "ground truth". As we note above, BreadCrumbs relies on PlaceLab [7] to estimate GPS location from observed WiFi beacons. The authors of PlaceLab found an average error of ±20 or 30 meters for their technique, as compared to GPS. Second, even if a GPS antenna is available on a mobile device, we must be mindful of the state-space explosion in the Markov model that would occur if a small grid size were chosen. BreadCrumbs is intended for small devices with limited storage, CPU and battery power, all of which would be taxed if maintaining a large mobility model.

The frequency with which BreadCrumbs estimates the device's GPS location bounds the resolution of the mobility model. This model can be thought of as a discrete-time Markov chain where a state transition fires every $\tau$ seconds. Figure 1 illustrates how the model generation process works. The first state is state 1. This is a special state with no "Last GPS" component, just the initial location. Then, $\tau$ seconds later BreadCrumbs fixes the device's location at $(80.276, -80.747)$, and creates the new state 2. The remaining states in the example are generated in a similar fashion.

For each state in the model, BreadCrumbs updates the Markov transition matrix whenever the model is in the state and transitions to another. These transitions occur every $\tau$ seconds. Note that if the user remains at one location for long periods, the model will have a heavy transition probability towards the self-loop (back to the same state) at that location. This is an easy way for BreadCrumbs to identify what others have termed *hubs* [10]—popular, long-term destinations.

## 3.2 Forecasting Future Conditions

Section 2.1 above described our prior work on determining the application-visible quality of WiFi access points. We use similar techniques here to build an AP quality database. The purpose of maintaining this database is to estimate the "quality" of a connection to the Internet, for all the different access points a mobile device encounters. As with Vir-

```
BBW (state x)
    best ← 0.00
    foreach ap ∈ {APs previously seen at state x}
        if ap.bandwidth > best
            best ← ap.bandwidth
    return best
```

(a) Best bandwidth algorithm

```
CF (state x_i, int steps)
    if steps ≤ 1
        return Σ_∀j {p_ij · BBW(x_j)}
    else
        return Σ_∀j {p_ij · CF(x_j, steps − 1)}
```

(b) Connectivity forecast algorithm

**Figure 2: Pseudocode: best bandwidth at a state and connectivity forecasts. The best bandwidth algorithm has been simplified to assume BreadCrumbs tracks one type of bandwidth, when in fact it differentiates between upstream and downstream connectivity.**

gil, when BreadCrumbs first encounters an unencrypted AP, it attempts to associate and obtain an IP address through DHCP. If successful, BreadCrumbs then opens three TCP connections to a remote reference server, to estimate (1) downstream bandwidth, (2) upstream bandwidth, and (3) latency to remote Internet hosts.

Building an AP quality database from scratch is admittedly taxing on mobile devices, given their limited battery life. In our prior work [21], we discuss how caching results in a local database hides this expense after an initial training period. As part of future work, we hope to deploy BreadCrumbs on the COPSE mobile device testbed[1] to investigate how sharing of these databases among co-located users can reduce this scanning overhead further.

If BreadCrumbs were broadly deployed and all users relied on the same reference server, the system would clearly not scale well. However, different users are free to use reference servers of their own choosing. BreadCrumbs is not attempting to quantify the quality of connection to a specific end host but rather to the more fuzzy notion of an arbitrary Internet destination.

It is true that one reference server cannot possibly represent the myriad network destinations that applications might contact. But note that the first hops—the wireless AP and its backend connection, e.g. a DSL or cable modem—are constant no matter what the remote destination of a connection ultimately is. From there, the path through the network core depends on the peering agreements between the AP's ISP and that of the destination. We argue that when choosing between two APs, it is far more likely that the overall quality of an end-to-end link depends on edge effects rather than core routing issues. This claim is validated by a recent measurement study [8] that found residential broadband links are overwhelmingly the bottleneck in end-to-end Internet paths.

A subtle point is that one access point may be visible from multiple grid locations, since our chosen grid size ($0.001^\circ \times 0.001^\circ$) is only 110m×80m at our latitude. The quality of an AP may vary at different grid locations, however, because of varying distances from the AP, physical interference, et cetera. BreadCrumbs therefore tags all AP test results with the GPS coordinates at which they were taken. Multiple test results for a single AP co-exist in the quality database if they were probed at different GPS grid locations.

The test database tracks access points both by ESSID and by MAC address. This is crucial to differentiate between APs sharing the same ESSID, either intentionally as part of a coordinated deployment or unintentionally because the default ESSID (e.g. `linksys, netgear`) has not been changed.

This test process incurs a reasonable but non-trivial overhead in terms of time and energy. BreadCrumbs therefore caches test results for performance. When an access point is detected, BreadCrumbs checks if a test results exists in the database for that AP at the GPS grid location containing the user's current position, and does not retest the AP if one exists. In order to age stale test results out of the database, however, BreadCrumbs retests such previously-probed APs probabilistically a small fraction of the time.

BreadCrumbs combines the custom user mobility model and the AP quality database to provide *connectivity forecasts*. Figure 2 describes a simplified version of this algorithm. This example takes two arguments: a state in the mobility model, and an integer number of steps in the future. In our actual implementation of BreadCrumbs, the algorithm also considers what network quality is to be forecast (downstream/upstream bandwidth, or latency). To simplify the pseudocode we assume the algorithm only considers one network quality metric, *bandwidth*.

First, consider the limiting case where *steps* is one. This is a request for the projected network bandwidth one transition past the specified state. In other words, for the model transition period $\tau$, one step is $\tau$ seconds in the future. BreadCrumbs calculates this forecast as the weighted sum, across all states in the model, of the best bandwidth previously seen from an AP at that potential next state. This sum is weighted by the transition probability that model will transition from state $x_i$ to a state $x_j$. Thus, the best bandwidth seen at states which are likely successors of the state contributes more to the connectivity forecast than transitions which are unlikely. In practice, the number of successor states from any given state will be small as compared to the whole state space, because states are grounded in geographic reality.

If *steps* is greater than one, connectivity forecasts are calculated recursively. At each step up the recursion tree, results from leaf nodes are weighted-summed in proportion to the transition probabilities.

## 3.3 Example

Consider the Markov chain in Figure 3. The value below each state's name is the best downstream bandwidth probed while at that state—for a state $x_i$, this is $\text{BBW}(x_i)$. The current state is $x_0$. We want to know the expected downstream bandwidth at the next time step. From Figure 2(b) above, this yields:

$$\text{CF}(x_0, 1) = \sum_{\forall j} p_{0j} \cdot \text{BBW}(x_j) \quad (1)$$

In other words, the expected downstream network bandwidth one step in the future is the sum (over all states in the Markov chain) of the best bandwidth observed at each

[1] `http://copse.cs.duke.edu/`

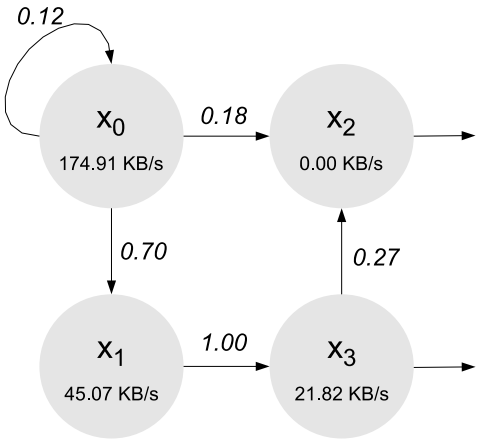



**Figure 3: Example Markov model with best-bandwidth results.**

state, weighted by the probability that the Markov chain will transition from the current state $x_0$ to each given state $x_j$. When calculating a connectivity forecast, we need not actually sum across all the states in the Markov chain, but only across those with a non-zero transition probability. Returning to our example, we see from Figure 3 that the only possible transitions out of state $x_0$ are to states $x_1$ and $x_2$, and a self-loop back to $x_0$. Therefore, Equation 1 above is simplified to:

$$\begin{aligned} \text{CF}(x_0, 1) &= p_{00} \cdot \text{BBW}(x_0) + p_{01} \cdot \text{BBW}(x_1) + p_{02} \cdot \text{BBW}(x_2) \\ &= 0.12 \cdot 174.91 + 0.70 \cdot 45.07 + 0.18 \cdot 0.00 \\ &= 52.54\ KB/s \end{aligned}$$

For instance, if the time step of the model was ten seconds, then this would be the estimated downstream network bandwidth available to the device ten seconds from the current time. To calculate connectivity forecasts further into the future, the connectivity forecast algorithm calls itself recursively as shown in Figure 2(b). The downstream bandwidth 20 seconds ahead (two steps) is therefore the following:

$$\begin{aligned} \text{CF}(x_0, 2) &= \sum_{\forall j} p_{0j} \cdot \text{CF}(x_j, 1) \\ &= p_{00} \cdot \text{CF}(x_0, 1) + p_{01} \cdot \text{CF}(x_1, 1) + p_{02} \cdot \text{CF}(x_2, 1) \end{aligned}$$

# 4. IMPLEMENTATION

We have implemented a BreadCrumbs prototype on Linux, as a user-level privileged process. This process consists of two threads, each of which is described in a subsection below.

## 4.1 Scanning Thread

One thread periodically scans for access points and fixes the device's GPS coordinates by triangulating on the locations of AP beacons in the Place Lab database. This scanning period is a configurable parameter ($\tau$), set to 10 seconds in our current implementation. The scanning thread also handles the probing of AP connection quality, as described in Section 2.1, whenever an open AP is encountered that has not been probed at the current GPS grid location. Test results are then stored in a local database.

After fixing its current GPS location every $\tau$ seconds, this thread then updates the Markov model. This consists of updating the transition probability from the previous state to the new current state (because of the new location estimate).

The reference server used to estimate AP connection quality was located on our university campus, connected directly to the Internet on the wired departmental network with no firewall. Given that our subsequent evaluation took place in the same city, one might be skeptical that connecting to this server from different wireless access points in the same city would truly approximate the average latency and bandwidth one would encounter when connecting to arbitrary remote destinations. The peering points between the university's ISP and the common ISPs seen around town—overwhelmingly, Comcast and AT&T—are not located in the city, however. In fact, for a subset of locations we performed a `traceroute` to the reference server, and in all cases the shortest path from the wireless AP to the departmental network detoured several hundred kilometers away, into the Internet core, before returning to our city. We are therefore confident that this configuration reasonably approximates the latency and bandwidth one would encounter when contacting typical Internet destinations that require a trip through the network core.

## 4.2 Application Interface

The other thread handles application requests for connectivity forecasts. Applications send requests to BreadCrumbs via a named pipe. These requests consist of two values: (1) the criterion of interest—downstream bandwidth, upstream bandwidth, or latency—and (2) an integer number of seconds in the future.

BreadCrumbs converts the value in seconds into the number of corresponding state transitions in the future of the model. This depends both on the scanning period $\tau$ and the number of seconds left until the start of the next scan, because the mobility model is a discrete time Markov chain where a state transition fires every $\tau$ seconds.

First, BreadCrumbs subtracts the time left until the start of the next scan from the value passed by the application. Then, it performs integer division of the remaining time by $\tau$. The result is the number of steps in the future of the model at which to generate a connectivity forecast.

For example, assume that BreadCrumbs scans for APs and updates the mobility model every 10 seconds (as in our implementation), starting at $t = 0$. At $t = 9$, an application queries for the forecasted downstream bandwidth 25 seconds in the future (at $t = 36$). This is $\lfloor (36 - 1)/10 \rfloor = 3$ steps in the future. BreadCrumbs then generates the connectivity forecast at that point in the future, for the given criterion, and returns the value to the calling application through the named pipe.

# 5. SAMPLE APPLICATIONS

In evaluating the usefulness of BreadCrumbs,we designed several simple applications that one might commonly find on mobile devices. We then examined how well BreadCrumbs can improve the user experience for these applications, as compared the best effort one could make without any connectivity forecast information.

The error bars in all subsequent figures in this section represent the standard error of the mean: $S_E = \sigma/\sqrt{n}$.

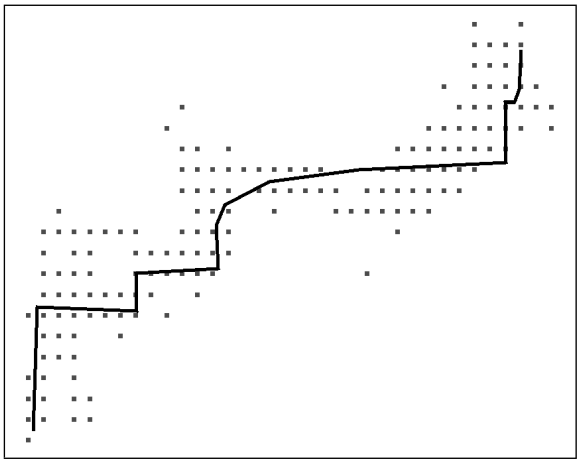


**Figure 4: Visited grid locations and *commute* ground truth. Small squares are all GPS grid locations fixes from two weeks of user mobility traces collected. The black line is the "ground truth" path through the map taken by the user on his daily commute between home and work.**

| | mean | $\sigma$ | max | min | n |
|---|---|---|---|---|---|
| **APs per scan** | 10.23 | 7.73 | 32 | 0 | 5227 |

| | |
|---|---|
| **unique APs** | 1621 |
| **open APs** | 282 (17.40%) |
| **encrypted APs** | 1339 (82.60%) |
| **grid locations visited** | 110 |
| **locations with usable AP** | 61 (55.45%) |

**Table 1: Access point statistics. *Locations with usable AP* are those grid locations where at least one access point had a probed downstream bandwidth greater than zero.**

| | mean | $\sigma$ | max | min | n |
|---|---|---|---|---|---|
| **down BW** | 68.38 | 114.41 | 385.54 | 0.00 | 110 |
| **down non-zero** | 123.30 | 129.74 | 385.54 | 0.29 | 61 |
| **up BW** | 33.98 | 49.85 | 241.66 | 0.00 | 110 |
| **up non-zero** | 64.44 | 52.44 | 241.66 | 4.10 | 58 |

**Table 2: Bandwidth at grid locations. Values in KB/s. According to Place Lab estimates, during the evaluation period the mobile device visited 110 unique grid locations (0.001° latitude by 0.001° longitude). *Non-zero* refers to omitting those locations where no encountered AP had a probed bandwidth greater than zero.**

## 5.1 Methodology

Rather than rely on existing mobility traces or synthetic models, we installed BreadCrumbs on an iPAQ h5555 handheld, with an integrated 802.11b WiFi card, running Familiar Linux (a distribution targeted for handheld devices [12]). One of the authors carried the handheld with him continuously for two weeks during daytime hours (before seven pm).

Clearly, most users are relatively stationary for large portions of their day—sitting at their desks at work, or moving around their home. Predicting connectivity in such situations is trivial. We were more concerned with how well BreadCrumbs predicts the derivative of connectivity when users are in motion. We therefore edited the collected logs by hand to remove portions of time when the user was stationary for more than five minutes.

BreadCrumbs ran continuously in the background, scanning for new access points every ten seconds. After each scan, BreadCrumbs estimated the device's current GPS coordinates by cross-referencing the MAC addresses of detected APs with the Place Lab database (as described in Section 2.2). The GPS coordinates and MAC addresses were then logged, along with a timestamp. For each AP in the scan set that had not been previously probed at those coordinates, BreadCrumbs attempted to associate and probe AP quality as described in Section 2.1. The probe results (upstream bandwidth, downstream bandwidth, latency) were then appended to a test results database.

Recall from Section 3.1 that BreadCrumbs divides the world into *grid locations*, where each grid box is 0.001° of latitude by 0.001° of longitude. At our latitude, this is 110 m×80 m. All GPS fixes that fall within the same box are considered to be the same position. The small squares in Figure 4 are all the unique grid locations visited during the two weeks of user traces. The solid black line represents the *ground truth path* of the user's daily commute between home and work. This trip is a mix of walking and bus riding, and is responsible for the vast majority of motion during the two week period. The spread of visited grid locations is not strictly limited to the commute path, however. This is a result both of Place Lab GPS error and noise introduced by other, non-commuting trips. For example, the trace set includes instances of the user walking from home to various downtown destinations, and driving to several different locations.

Tables 1 and 2 summarize the frequency and quality of network connectivity that BreadCrumbs encountered during the course of our evaluation. As Table 1 shows, BreadCrumbs saw a widely-varying number of APs each time it scanned. While only 17% of all access points encountered were unencrypted, BreadCrumbs was able to discover a usable AP at over half of all visited grid locations. We define *usable* to mean there existed an AP at that location whose probed downstream bandwidth was greater than zero.

As Table 2 shows, we found that the quality of publicly-available access points varies significantly. For each of the 110 grid locations visited during the two weeks of trace collection, we calculated the best upstream and downstream bandwidth available. Even when those locations where no AP had a non-zero bandwidth are omitted, the variance is quite large. This bolsters our claim that network connectivity fluctuates significantly as users move around the world.

## 5.2 Forecast Accuracy

We first wanted to quantify how accurate connectivity forecasts are, given the two weeks of traces we collected. As a reminder, BreadCrumbs estimates its GPS coordinates at a fixed frequency. For our evaluation we set this period to ten seconds. Thus, the traces are a series of scan sets—listing all AP beacons detected, plus current GPS coordinates and a timestamp—separated by ten seconds of real time.

We used the first week of traces as the training set that built BreadCrumbs' mobility model. The second week of traces was then the evaluation set. For each step (scan set) in the evaluation set of traces, we compared the grid location where BreadCrumbs predicted the device would be in the next step with where it actually did move. We repeated this,

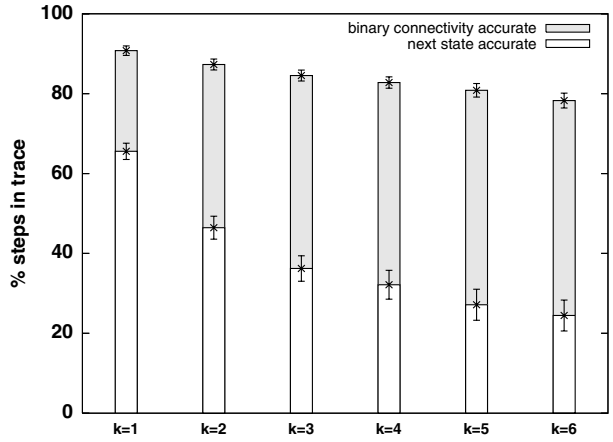



**Figure 5: Mobility model prediction accuracy. $k$ indicates the number of steps into the future BreadCrumbs forecasts.**

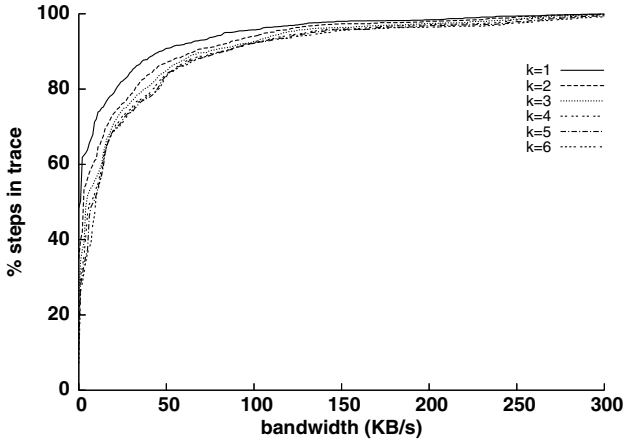



**Figure 6: CDF: bandwidth prediction error. $k$ indicates the number of steps into the future BreadCrumbs forecasts.**

varying the number of steps BreadCrumbs looked ahead ($k$) from one through six. The white bars in Figure 5 indicate the percentage of steps across all two weeks of traces where BreadCrumbs' predicted grid location was correct, for $1 \leq k \leq 6$. The accuracy is over 70% for $k = 1$ but quickly degrades as BreadCrumbs must extrapolate further into the future.

The crucial insight, however, is that we are not really concerned with predicting the user's mobility perfectly. If BreadCrumbs predicts the user will move to one location, and they in fact move to another, as long as the quality of network connectivity available at the two locations is comparable this "mistake" is unimportant. The gray bars in Figure 5 represent the percentage of steps where BreadCrumbs' prediction and the actual next location matched with regard to binary connectivity. A given location is considered *connected* if at least one AP seen at that location had a probed downstream bandwidth greater than zero. BreadCrumbs was over 90% accurate in predicting binary connectivity one step ahead. This accuracy remained high when looking further into the future—nearly 80% accurate six steps ahead.

Next, we examined how the bandwidth predicted by connectivity forecasts matched the bandwidth actually encountered. Figure 6 charts the gap between predicted and actual bandwidth as a cumulative distribution function (CDF). Even six steps in the future, BreadCrumbs' bandwidth forecasts were within 10 KB/s of the actual value for over 50% of the trace period, and within 50 KB/s for over 80%.

It is important to note that these results were achieved with a training set of only one week duration. As users run BreadCrumbs for increasingly-long periods, the device-centric mobility model can only benefit from increased exposure to the user's patterns.

## 5.3 Sample Applications

The primary aim of BreadCrumbs is to improve the application-level and (most importantly) user-visible experience for mobile devices. To truly evaluate our system, then, we need to examine how both the operating system and different mobile applications could benefit from connectivity forecasts.

We evaluate the performance of different applications using the traces we collected, rather than executing the applications "live" on a mobile device. This allows us to directly compare the performance of prediction-unaware algorithms and BreadCrumbs on identical sequences of user motion and APs seen, to ensure an accurate comparison.

The subsections that follow investigate three such scenarios. Clearly, connectivity forecasts are most useful for background or opportunistic tasks, where an application has some flexibility in when a network operation must occur.

As in Section 5.2, the first week of traces was the training set that built the mobility model, and the second week the evaluation set. For each scenario we devised three algorithms that accomplished the same objective—one that was ignorant of any future predictions, another that utilized BreadCrumbs' connectivity forecasts, and a third that used a random walk mobility model. For each trace in the evaluation set, we ran all three algorithms, recorded the results, and subsequently averaged across all the runs. A "step" in each trace corresponds to 10 seconds of real time.

At each step in the trace, for all algorithms, the simulation declared the device associated to the AP with the best downstream bandwidth among all APs present that that location. This corresponds to the device using the aforementioned Virgil AP selection system [21] to choose the current AP, rather than simply selecting based on signal strength. The *No Prediction* algorithm therefore represents the best one could do making no predictions of future connectivity, but using the best AP available a the current location for each step.

### 5.3.1 Map Viewer

Our first sample application is a map viewer, commonly found on mobile devices like the Nokia N800. This application displays a map of the user's current location, and is typically linked to a GPS receiver so as the user moves, the currently-displayed map tile is updated to reflect this movement. Beyond simple street maps, these map tiles can contain rich contextual information—such as menus and reviews of nearby restaurants—or detailed geographic information, such as provided by Google Earth.

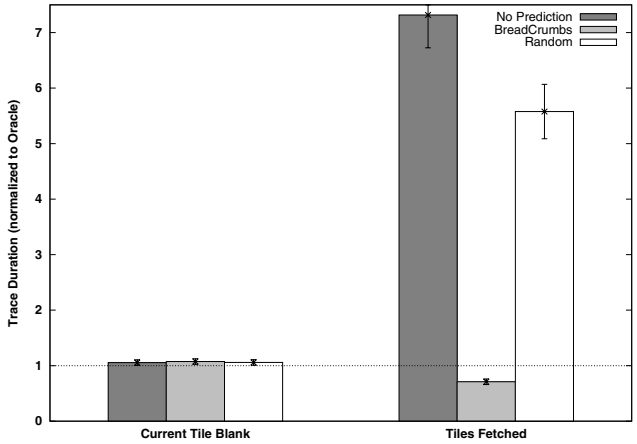



**Figure 7: Map Viewer.** ***Current Tile Blank*** **are steps in the trace where the tile corresponding to the current GPS location is not present in the device's cache, and insufficient network bandwidth exists to download it synchronously.** ***Tiles Fetched*** **is the total number of map tiles fetched over the course of each trace. All values are normalized to those of an Oracle algorithm that uses perfect knowledge of future mobility to minimize** ***Current Tile Blank.*** **The** ***BreadCrumbs*** **algorithm avoids unnecessary network traffic by not pre-fetching neighboring tiles when upcoming network conditions are predicted to be good, while incurring a slightly higher rate of missing tiles.**

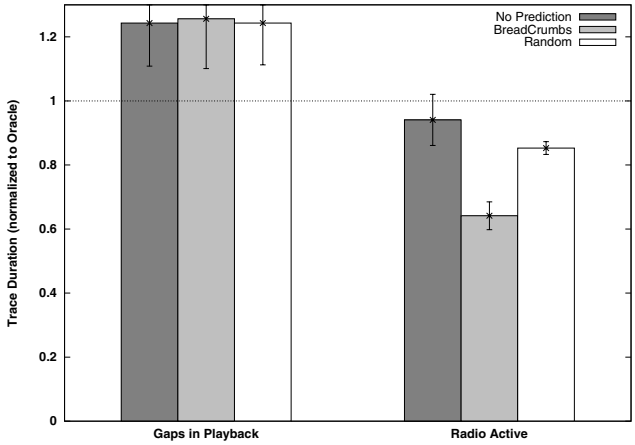



**Figure 8: Streaming Media.** ***Gaps in Playback*** **is the trace duration where the stream was not playing on the device because the buffer was empty and inadequate network connectivity existed at that location.** ***Radio Active*** **is the trace duration that the WiFi radio was actively downloading data. All values are normalized to those of an Oracle algorithm that minimizes** ***Gaps in Playback*** **by downloading the entire stream as fast as possible with an infinite buffer size. The** ***BreadCrumbs*** **algorithm avoids pre-filling the buffer if forecasts indicate that upcoming network bandwidth will be sufficient to service the stream. This conserves energy while not significantly increasing playback gaps.**

When the user moves out of one map tile and into another, the tile's information must either be fetched synchronously or already be present in a cache. Otherwise the user experience degrades as blank tiles appear in the map. A policy of always pre-fetching all neighboring map tiles provides good coverage, but at the cost of wasted network operations if those tiles are never visited or displayed.

We investigated if BreadCrumbs' forecasts could be used to "roll the dice" and avoid wasteful pre-fetching in cases where BreadCrumbs predicted that the device will have sufficient network bandwidth available to synchronously fetch a new map tile as soon as the user moves to that location. We therefore designed four algorithms for comparison.

First, *No Prediction* simply ensures that the user's current map tile, and all eight surrounding tiles, are present in the cache whenever sufficient network bandwidth exists to do so. Second, at each step in each trace, the *BreadCrumbs* algorithm generates a connectivity forecast one step in the future. If the forecast indicates that the mobile device will have enough bandwidth available to synchronously download its new map tile, the *BreadCrumbs* algorithm does not pre-fetch neighboring blocks. If the predicted next-step bandwidth is low, however, it pre-fetches neighboring blocks just as *No Prediction*. Third, the *Random* algorithm is identical to *BreadCrumbs*, but instead of using BreadCrumbs' connectivity forecasts, this algorithm chooses a random successor state to the current state, and takes the best bandwidth observed at that state as the bandwidth the device will have at the next step in the trace.

Finally, the *Oracle* algorithm uses perfect knowledge of future mobility to achieve the minimum possible number of blank tiles per trace. Note that minimizing one criterion (*Current Tile Blank*) does not necessarily optimize for the other (*Tiles Fetched*). In fact, we will see that the *BreadCrumbs* algorithm fetches fewer tiles than the *Oracle* because it risks blank tiles in order to fetch as few tiles as possible from the remote server.

Tiles correspond to the 110×80 meter tiles of our model, and each tile is assumed to be 100 KB in size. The results in Figure 7 are all normalized to those of the *Oracle* algorithm. One sees that by not pre-fetching neighboring tiles when upcoming connectivity is predicted to be good, the *BreadCrumbs* algorithm avoids wasting energy by fetching tiles that will never be displayed. At the same time, this gamble results in only a three percent higher rate of missing map tiles than the *No Prediction* algorithm.

### 5.3.2 Streaming Media

Next, we considered issues raised when streaming media content from a remote server onto a handheld device while the user is in motion. A media stream has a well-defined quality-of-service metric—specifically, the encoded bit rate of the stream. When mobile in public, however, the user's device moves from connection to connection at different locations. Some locations may have sufficient bandwidth to service the stream, some may not, and some locations may be devoid of network connectivity altogether.

One option is to define a buffer size and fetch the stream as fast as possible at every given moment, up to the point where the buffer is filled. This is the strategy commonly employed today by streaming media applications, corresponding to the *No Prediction* algorithm for this application. This algorithm

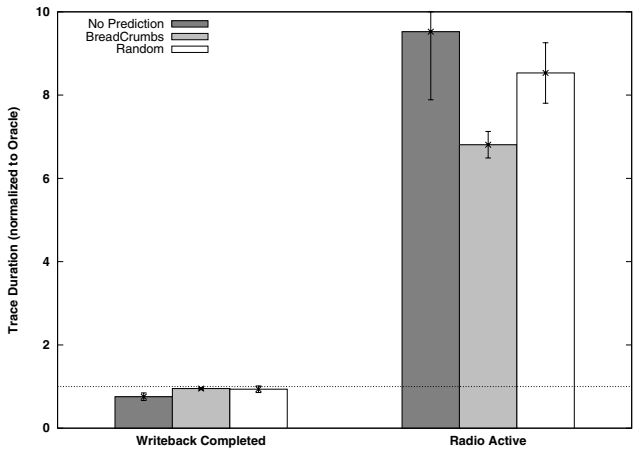



**Figure 9: Opportunistic writeback. *Writeback Completed* is the total elapsed trace time until all data was safe on the remote server. *Radio Active* is the total trace time that the WiFi radio was actively transmitting data. All values normalized against an Oracle algorithm that uses perfect knowledge of future mobility to minimize *Radio Active*. By utilizing BreadCrumbs' connectivity forecasts, the prediction-aware algorithm delays data writeback briefly to selectively use high-bandwidth access points. As a result, the total time until data is safe on the remote server is comparable, but BreadCrumbs activates the WiFi radio 30% less often than with no prediction, translating into significant energy savings.**

downloads the stream as fast as possible at each step in the trace, given the available network connectivity, up until the point that a two-minute buffer has been filled.

Second, at each step the *BreadCrumbs* algorithm generates connectivity forecasts for each of the next six steps—up to one minute in the future. If the future connectivity is predicted to be sufficient to service the media stream, the algorithm does not pre-fetch data into the buffer. The *Random* algorithm is identical, except that instead of using connectivity forecasts to predict future network conditions, it generates a random walk from the current location into the future.

Finally, the *Oracle* algorithm downloads the entire stream as fast as possible, without a buffer size cap. This is less of an "oracle" than a bound on how quickly any algorithm could fetch the data comprising the stream for the trace duration. Note that while this minimizes *Gaps in Playback* it results in the radio being active for longer than any of the other algorithms.

We simulated a 64 KB/s video stream. Figure 8 shows that all three algorithms (*No Prediction*, *BreadCrumbs*, and *Random*) result in comparable gaps in playback. The *BreadCrumbs* algorithm, however, activates the WiFi radio 30% less often than the prediction-unaware algorithm. By employing BreadCrumbs' connectivity forecasts, that algorithm is able to provide the same playback experience to the user while using significantly less of the mobile device's battery as compared to a prediction-ignorant algorithm.

### 5.3.3 Opportunistic Writeback

Our final scenario considers a user who has generated some content on his handheld device while away from home. These files are digital photos taken by the camera on his smartphone. The user previously configured a distributed file system client to ensure all content he generates will be safely reintegrated to his remote file server. This file server could be a dedicated machine at his home or work, or a web service such as Flickr. We assume the only network connectivity available to the smartphone is whatever open WiFi is available.

For evaluation purposes, we set the number of photos that our hypothetical user took at eight, each with a filesize randomly uniform between 1 MB and 5 MB. The set of eight random filesizes was generated once and then the same set used across the entire evaluation for consistency.

The *No Prediction* algorithm simply tried to transmit the eight image files as quickly as possible, at each step using the AP with the best upstream bandwidth available at that location. The algorithm that utilized BreadCrumbs sought to reduce the amount of time the WiFi radio was active, while not delaying data writeback unreasonably. Our simple prediction-aware algorithm worked as follows. At each step of trace playback:

1. Determine which AP has the best upstream bandwidth at the current location.
2. Query BreadCrumbs for its connectivity forecast of upstream bandwidth 10, 20, and 30 seconds in the future. If any of those three future points are predicted to have better upstream bandwidth, do nothing at this time. Else, transmit data to the remote server as fast as possible during this step.

This algorithm is admittedly somewhat naïve. This was intentional as we sought to evaluate how useful BreadCrumbs' connectivity forecasts could be for applications that have made very minimal modifications. A third algorithm, *Random*, operated the same as the BreadCrumbs algorithm but instead of using connectivity forecasts to predict future network conditions, *Random* simply generated a random walk through the geographic neighbors of a given state in order to "predict" future mobility and connectivity.

Finally, a fourth algorithm, *Oracle*, is our baseline. This algorithm uses perfect knowledge of the future to generate the minimal radio active time possible, by only transmitting data at the states of each trace with the highest available AP bandwidth.

We ran all four algorithms once for each of the traces in the evaluation set. Our evaluation metrics were (1) total elapsed time until the all data was safely on the remote server, and (2) total time the WiFi radio was actively transmitting. Figure 9 illustrates the results. On average, the BreadCrumbs algorithm completes writeback only slightly slower than the aggressive, prediction-ignorant algorithm. In fact the difference is nearly within the error bounds of the mean for both algorithms.

On the other hand, utilizing BreadCrumbs' connectivity forecasts lets the prediction-aware algorithm activate the WiFi radio 30% less often. By attempting to only transmit data at high-bandwidth locations, the prediction-aware algorithm makes more efficient use of the wireless radio. While small for desktops or even laptops, this is significant for mobile devices where wireless NIC usage is a large fraction of

| # states in model | 652 |
|---|---|
| model size | 27984 bytes (42.92 B/state) |
| # test results | 1335 |
| test DB size | 92132 bytes (69.01 B/entry) |

**Table 3: Overhead: space requirements. The test database is currently stored in unoptimized, ASCII format.**

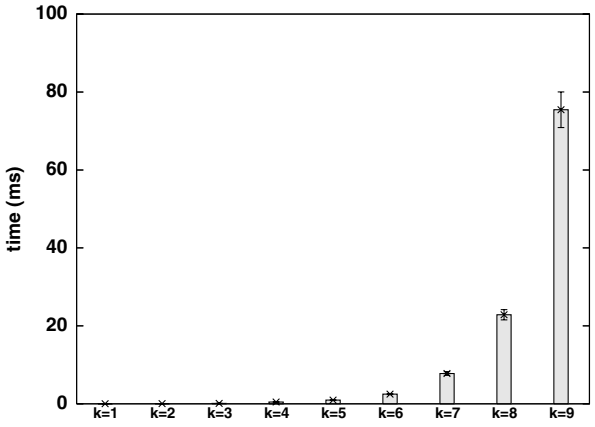



**Figure 10: Connectivity forecast overhead. Results on a Compaq iPAQ handheld (400 MHz CPU), 128 MB RAM.**

total energy expenditure. For example, Anand et al [3] found that, for an iPAQ handheld, the power required to actively transmit data over the WiFi interface (even in power-save mode) was nearly equal to the measured quiescent power consumption of the entire device when the radio was inactive.

## 5.4 Overhead

Table 3 shows the storage required on the iPAQ to store the mobility model and test database generated in the course of our evaluation. With 652 different states in the model, the total model size is approximately 27.3 KB, or 43 bytes per state on average. Recall that, because ours is a second-order Markov model, each state represents the current GPS grid location of the user and their previous location. From Table 1, we know that BreadCrumbs visited 110 different grid locations during the evaluation period. If every combination of current location and previous location were generated as a state, the model would have $110 \times 110 = 12100$ states. Even a model of such complexity would only require 508 KB of space on the mobile device. Given the sparseness of these models in practice, a model of that size would be most likely be sufficient to cover an entire metropolitan area.

Likewise, the overhead imposed to store the test database is reasonable—69 bytes per test entry on average. For convenience, the database was implemented as an ASCII flat file, unoptimized. Even so, the records for the 1335 test results generated by our evaluation require 90 KB of storage space, but only 7.04 KB when in compressed form.

Figure 10 examines the CPU overhead imposed when generating connectivity forecasts. The parameter $k$ is the number of steps in the future of the model, given a current state, that we requested a connectivity forecast of downstream bandwidth from BreadCrumbs. This graph represents only the instrumented CPU time required for the calculation, not any communications overhead between BreadCrumbs and the application requesting the forecast. All results were measured on a Compaq iPAQ h5555, with a 400 MHz ARM processor and 128 MB of system RAM.

We requested a connectivity forecast for each of the 652 states in the model our evaluation generated, varying the size of $k$ from 1 to 10. Because this is a recursive algorithm (see Figure 2) we expect the overhead to grow exponentially. Up to six steps ahead, the overhead is less than 2.5 ms. Even the mean overhead of 75 ms at $k = 9$ is not prohibitive for applications that perform such intensive operations rarely. Note that we did not implement caching of calculated forecasts or other possible optimizations in our implementation.

# 6. RELATED WORK

Rahmati and Zhong [23] investigated the problem of choosing between WiFi and cellular data networks, given that a large number of mobile devices now feature both radios (e.g. the Apple iPhone). Rather than build and maintain a mobility model as BreadCrumbs does, they use the set of cellular tower IDs currently seen and some time-of-day heuristics to estimate the expected quality of WiFi connectivity at the current location. Their system does not predict future conditions, as BreadCrumbs does. Rather, it decides whether at a given time and place, it is more advantageous to power on the WiFi interface or to use the lower-bandwidth, but ubiquitious, cellular connection. Also, identifying location solely by cell tower signals is necessarily more coarse-grained than our approach that leverages WiFi beacons or GPS. Cellular signals reach at least several kilometers, and tens of kilometers under good conditions. WiFi signals have a far shorter range, typically several hundred meters at best.

MobiSteer [20] focuses on improving wireless network connectivity in one specific usage setting—while in motion in a motor vehicle. Their system uses a directional antenna to maximize the duration and quality of connectivity between a moving vehicle and stationary access points in the community. This goal is complementary to that of BreadCrumbs, because MobiSteer performs well in situations where BreadCrumbs does not. While portions of the evaluation traces collected in our paper track the user riding on a city bus, during this period the user only has reliable connectivity while stopped at intersections. As explored in detail by Bychkovsky et al [5], this reduced performance was due to the brief time the client has to associate with the AP, obtain a DHCP address, and do useful work. On the other hand, BreadCrumbs does not require any specialized hardware and works with whatever users already carry in their pocket. MobiSteer's cached mode operation is also reminiscent of the way BreadCrumbs and Virgil [21] optimize future resource discovery by caching historical access point quality information.

Song et al [25] applied different mobility prediction methods to the problem of improving bandwidth provisioning and handoff for VoIP telephony. They use real client traces to evaluate the success of a concrete application that is prediction-aware, and assume the existence of a centralized authority that collects mobility information, makes predictions, and disseminates instructions to various wireless access points. We are focused on applications that are still useful when the device keeps its mobility history, and this information need not be disclosed to any other party.

Ghosh et al [10] predict the probability that users visit popular locations, known as *hubs*. Their focus is on extrapolating *sociological orbits* from the client mobility data by identifying the frequency with which users encounter one another at these hubs. The authors do not evaluate how accurately their Bayesian techniques predicted explicit client paths (rather than just the hubs they visit). We therefore were unable to compare the accuracy of their technique with that of our second-order Markov model.

Yoon [28] concentrated, as did Kim et al [14], on deriving realistic mobility models from actual user mobility traces. The idea is to take many different client traces and build a probabilistic model that can be used to generate arbitrary client tracks. These traces, while still artificial, more closely model the real movements of users than do synthetic models like Random Waypoint [27]. In this paper, we consider only the situation where devices maintain their actual mobility history themselves, and predict their future behavior "on-the-fly" rather than base predictions on mobility models derived from multiple users' behavior.

Marmasse [19] argues, as we do, in favor of a user-centric mobility model. Her *comMotion* system is concerned chiefly with tracking users' movement through various semantically meaningful locations, such as "home" or "work". We, on the other hand, focus on lower-level waypoints—namely, GPS grid locations. The semantic concept of user-defined locations could easily be layered atop such low-level information, however.

Haggle [13] is a framework for disseminating data between mobile users based on the fleeting occasions when they come into physical contact with each other. In these situations infrastructure such as WiFi networks need not be used, because users are within range of low-power, point-to-point link technologies like Bluetooth or ZigBee. Their system is clearly dependent on user-centric mobility information, but seeks to predict when pairs of users will come into contact with each other. Our work, on the other hand, is focused more on leveraging information about wireless access points the user will soon encounter.

Most applications of location prediction have been in mobile phone networks. Typically, a central network operator seeks to know the sequence of network towers with which a handset will associate. Given this information, the network operator can reserve resources, such as bandwidth, at the upcoming nodes, so handoff proceeds as smoothly as possible. Bhattacharya and Das [4] use a variant of the LZ predictor described above to predict the next cell users will associate with. Yu and Leung [29] extend this idea to predict not only where a mobile device will hand off but also when this will occur. Liang and Haas [17] use a Gauss-Markov model in a similar way. Others use Robust Extended Kalman Filtering (REKF) [22], integrate individual path information with system-wide aggregate data [1], or estimate future locations through trajectory analysis [2]. Liu et al [18] use a similar hybrid approach for mobility prediction in wireless ATM networks, rather than for mobile telephony. They combine system-wide information with local mobility history and path trajectories to reduce system resource consumption while maintaining user QoS.

All of these location predictors are enabled by accurate estimates of a mobile device's location. In some cases, all that is needed is information on which access point or mobile phone tower the device is associated with. For predictors and applications requiring more fine grained location information, there are a wide variety of solutions. Place Lab leverages public *war-driving* databases of WiFi AP GPS coordinates to triangulate one's location based on the APs seen at a given location and their signal strengths [16]. The same idea has recently been extended to use GSM phone towers rather than WiFi APs [6]. Fox et al [9] showed the benefit of Bayesian filtering to coalesce results from multiple location sensors and smooth transient uncertainty in location estimates. Other work focuses on indoor localization at very small scales, either by deploying custom hardware [24] or mapping existing WiFi beacon sources [11].

# 7. CONCLUSION

Operating systems currently focus on immediate conditions when managing wireless network connections. But today, users are more mobile than ever, utilizing a patchwork of public access points of varying capabilities and uneven geographic distribution. Applications would like to use this public connectivity opportunistically to perform background or low-priority work, but cannot make reliable assumptions about connection quality at any given moment in the future.

We argue that the increased mobility of users demands a focus on how connectivity changes over time—its *derivative*. This paper described BreadCrumbs, our system that let a mobile device track this trend of connectivity quality as its owner moves around the world. BreadCrumbs maintains a personalized mobility history on the device, and tracks the APs encountered at different locations. BreadCrumbs also probes the application-level quality—bandwidth and latency to the Internet—of the open connections the device encounters.

Together, the predictions of the mobility model and the AP quality database yield *connectivity forecasts*. These forecasts let applications take domain-specific action in response to upcoming network conditions. We evaluated the efficacy of these forecasts with several weeks of real-world usage. BreadCrumbs was able to predict downstream bandwidth at the next step of the model within 10 KB/s for over 50% of the evaluation period, and within 50 KB/s for over 80% of the time, with only one week of training data to build the model and AP quality database. We also evaluated how three example applications, with minimal modification, can utilize connectivity forecasts. Our results found that with as little as one week training time, BreadCrumbs can provide improved performance while reducing power consumption, a critical concern for resource-constrained mobile devices.

# 8. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under award number CNS-0615086, and the Ford Motor Company. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation or Ford Motor Company.

We would like to thank the anonymous reviewers, for their insightful comments and suggestions that greatly improved the quality of our paper. We also gratefully acknowledge the helpful feedback of Jason Flinn, Mingyan Liu, Z. Morley Mao, James Mickens, Sam Shah and Kaushik Veeraraghavan.

# 9. REFERENCES


[1] I. Akyildiz and W. Wang. The predictive user mobility profile framework for wireless multimedia networks. *IEEE/ACM Transactions on Networking*, 12(6):1021–1035, 2004.

[2] A. Aljadhai and T. Znati. Predictive mobility support for QoS provisioning in mobile wireless environments. *IEEE Journal on Selected Areas in Communications*, 19(10):1915–1930, October 2001.

[3] M. Anand, E. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *Proceedings of MobiCom*, pages 176–189, September 2003.

[4] A. Bhattacharya and S. Das. Lezi-update: an information-theoretic approach to track mobile users in PCS networks. In *Proceedings of Mobicom*, pages 1–12, 1999.

[5] V. Bychkovsky, B. Hull, A. Miu, H. Balakrishnan, and S. Madden. A measurement study of vehicular internet access using in situ Wi-Fi networks. In *Proceedings of MobiCom*, 2006.

[6] M. Chen, T. Sohn, D. Chmelev, D. Haehnel, J. Hightower, J. Hughes, A. LaMarca, F. Potter, I. Smith, and A. Varshavsky. Practical metropolitan-scale positioning for GSM phones. In *Proceedings of UbiComp*, pages 225–242, September 2006.

[7] Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. Accuracy characterization for metropolitan-scale Wi-Fi localization. In *Proceedings of MobiSys*, pages 233–245, June 2005.

[8] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroui. Characterizing residential broadband networks. In *Proceedings of IMC*, October 2007.

[9] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July–September 2003.

[10] J. Ghosh, M. Beal, H. Ngo, and C. Qiao. On profiling mobility and predicting locations of campus-wide wireless network users. In *Proceedings of REALMAN*, pages 55–62, May 2006.

[11] A. Haeberlen, E. Flannery, A. Ladd, A. Rudys, D. Wallach, and L. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of MobiCom*, pages 70–84, 2004.

[12] Familiar Linux. `http://familiar.handhelds.org/`.

[13] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, pages 244–251, August 2005.

[14] M. Kim, D. Kotz, and S. Kim. Extracting a mobility model from real user traces. In *Proceedings of INFOCOM*, April 2006.

[15] D. Kotz, T. Henderson, and I. Abyzov. CRAWDAD trace set dartmouth/campus/movement (v. 2005-03-08), Mar. 2005.

[16] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place Lab: Device positioning using radio beacons in the wild. In *Procedings of Pervasive*, pages 116–133, May 2005.

[17] B. Liang and Z. Haas. Predictive distance-based mobility management for multidimensional PCS networks. *IEEE/ACM Transactions on Networking*, 11(5):718–732, October 2003.

[18] T. Liu, P. Bahl, and I. Chlamtac. Mobility modelling, location tracking, and trajectory prediction in wireless ATM networks. *IEEE Journal on Selected Areas in Communications*, 16(6):922–936, August 1998.

[19] N. Marmasse and C. Schmandt. A user-centered location model. *Personal and Ubiquitous Computing*, 6(5–6):318–321, December 2002.

[20] V. Navda, A. Subramanian, K. Dhanasekaran, A. Timm-Giel, and S. Das. MobiSteer: Using steerable beam directional antenna for vehicular network access. In *Proceedings of MobiSys*, June 2007.

[21] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, and D. Wetherall. Improved access point selection. In *Proceedings of MobiSys*, pages 233–245, June 2006.

[22] P. Pathirana, A. Savkin, and S. Jha. Mobility modelling and trajectory prediction for cellular networks with mobile base stations. In *Proceedings of MobiHoc*, pages 213–221, 2003.

[23] A. Rahmati and L. Zhong. Context-for-wireless: Context-sensitive energy-efficient wireless data transfer. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications and Systems (MobiSys '07)*, pages 165–178, San Juan, Puerto Rico, June 2007.

[24] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha. Tracking moving devices with the Cricket location system. In *Proceedings of MobiSys*, pages 190–202, 2004.

[25] L. Song, U. Deshpande, U. Kozat, D. Kotz, and R. Jain. Predictability of WLAN mobility and its effects on bandwidth provisioning. In *Proceedings of INFOCOM*, April 2006.

[26] L. Song, D. Kotz, R. Jain, and X. He. Evaluating location predictors with extensive Wi-Fi mobility data. In *Proceedings of INFOCOM*, pages 1414–1424, March 2004.

[27] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *Proceedings of INFOCOM*, pages 1312–1321, March 2003.

[28] J. Yoon, B. Noble, M. Liu, and M. Kim. Building realistic mobility models from coarse-grained traces. In *Proceedings of MobiSys*, pages 177–190, June 2006.

[29] F. Yu and V. Leung. Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks. In *Proceedings of INFOCOM*, pages 518–526, April 2001.