BreadCrumbs: Forecasting Mobile Connectivity

Anthony J. Nicholson and Brian D. Noble
Department of Electrical Engineering and Computer Science
University of Michigan
{tonynich,bnoble}@umich.edu

Abstract

As mobile devices continue to shrink, users are no longer merely nomads, but truly mobile, employing devices on the move. At the same time, these users no longer rely on a single managed network, but exploit a wide variety of connectivity options as they spend their day. Together, these trends argue that systems must consider the *derivative* of connectivity—the changes inherent in movement between separately managed networks, with widely varying capabilities.

To manage the derivative of connectivity, we exploit the fact 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. Rather than rely on a synthetic model or aggregate observations, this custom-tailored model can be used together with past observations of wireless network capabilities to generate *connectivity forecasts*. Applications can in turn use these forecasts to plan future network use with confidence. We have built a BreadCrumbs prototype, and evaluated it 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.

1 Introduction

Operating systems manage wireless networks only *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. However, true mobile usage across a diverse and separately-managed patchwork of connectivity options presents both new challenges and opportunities.

In such an environment, mobile devices can work opportunistically as the they encounter public network connectivity. However, applications cannot make reliable assumptions about the quality of network connection at any given moment. With mobility, available network connectivity fluctuates, depending both on the path taken through uncoordinated public deployments and the varied quality of individual access points.

Reactive management performs poorly for this collection of always-active, mobile devices attempting to use a patchwork of uneven, unmanaged connectivity. 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, predicting future network conditions based past movements. Because people are creatures of habit, the predictions of this model are highly accurate with even minimal training time. BreadCrumbs tracks not only which APs were seen at a location, but also probes their application-visible quality.

BreadCrumbs combines the predictions of its mobility model with an AP quality database to produce *connectivity forecasts*. Applications, or the operating system itself, use these forecasts to take domain-specific actions in response to upcoming network conditions, deferring less time-sensitive or low-priority work to a time that will maximize performance or minimize power consumption.

We evaluated our prototype system with several weeks of real-world usage in Ann Arbor, Michigan. We found that both the quality and the availability of publicly-accessible APs are 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 evaluated how BreadCrumbs' connectivity forecasts could aid three example applications: (1) opportunistic writeback of created media content, (2) deactivating a wireless link in an area without coverage to save power, and (3) assigning data flows between WiFi and cellular data connections. Compared to prediction-ignorant baselines, BreadCrumbs' forecasts improve power efficiency and application performance for the first two applications, while the results for the third are somewhat mixed.

This work makes the following contributions:

- We introduce the concept of *connectivity forecasts* and show how they are useful to realistic applications on always-active, mobile devices.
- We show that such devices can predict not just their future mobility patterns but quality of upcoming network connectivity with high accuracy, without requiring GPS hardware or extensive centralized infrastructure.
- We evaluate our system using the actual mobility of a real person as he encountered the *in situ* deployment of access points in a city.
- We show how resource-constrained mobile devices can do this efficiently, respecting both CPU and storage limitations.

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 [25], 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. 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 Angell Hall”), 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 [20]. 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.

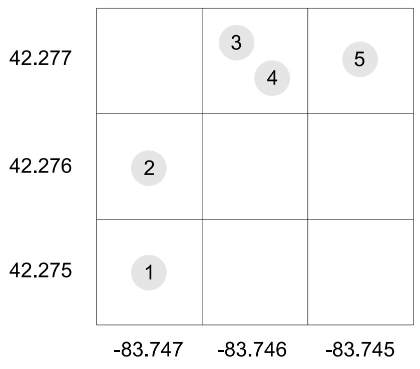
This section first discusses how BreadCrumbs maintains a personalized device mobility model, based on the past sequence of GPS locations the user visits. Next, we describe how BreadCrumbs applies the principles of Virgil [25] to estimate the quality of different access points, and combines this data with the predictions of the mobility model. The section concludes with a concrete example of how connectivity forecasts are generated.

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 [2, 3, 7, 21, 26, 28, 32]. 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 [30] or aggregate models derived from the movements of many users [18, 31] 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 mobility model on the device itself is that, unlike for a mobile phone network,



State	Last GPS	Current GPS
1	—	(42.275,-83.747)
2	(42.275,-83.747)	(42.276,-83.747)
3	(42.276,-83.747)	(42.277,-83.746)
4	(42.277,-83.746)	(42.277,-83.746)
5	(42.277,-83.746)	(42.277,-83.745)

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 τ that is the time interval between state transitions in the model.

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 [29] 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 [19]. 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 distinguish between the user walking eastbound and westbound on a given 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 in Ann Arbor, Michigan (42.2°N), a $0.001^\circ \times 0.001^\circ$ grid square is 110 m×80 m.

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 τ 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, τ seconds later BreadCrumbs fixes the device's location at $(42.276, -83.747)$, 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 τ 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* [13]—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 Virgil, when BreadCrumbs first encounters an unencrypted AP, it attempts to associate and obtain an IP address through DHCP. If successful, BreadCrumbs then opens three connections to a remote reference server, to estimate (1) downstream bandwidth, (2) upstream bandwidth, and (3) latency to remote Internet hosts.

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.

```
BBW (state $x$)
    $best \leftarrow 0.00$
    **foreach** $ap \in$ {APs previously seen at state $x$}
        **if** $ap.bandwidth > best$
            $best \leftarrow ap.bandwidth$
    **return** $best$
```

(a) Best bandwidth algorithm

```
CF (state $x_i$, int $steps$)
    **if** $steps \leq 1$
        **return** $\sum_{\forall j}\{p_{ij} \cdot \mathrm{BBW}(x_j)\}$
    **else**
        **return** $\sum_{\forall j}\{p_{ij} \cdot \mathrm{CF}(x_j, steps - 1)\}$
```

(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.

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 Ann Arbor's 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.

BreadCrumbs combines the custom user mobility model and the AP quality database to provide *connectivity forecasts*. Figure 2(b) 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 τ, one step is τ 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

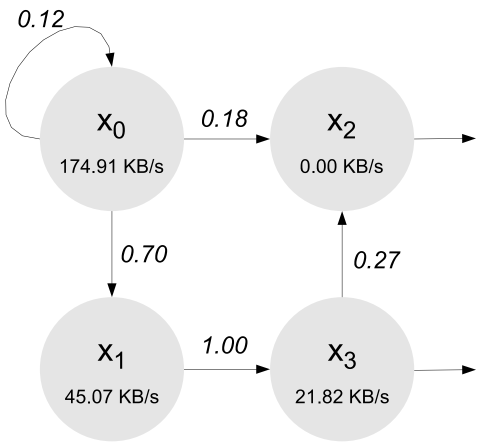


Figure 3: **Example Markov model with best-bandwidth results**.

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 as shown in Figure 2(b). 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 $\mathrm{BBW}(x_i)$. 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:

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

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 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 nonzero 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:

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

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:

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

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 (τ), 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 τ 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 the campus of the University of Michigan, connected directly to the Internet on the wired EECS network with no firewall. Given that our subsequent evaluation took place in Ann Arbor, 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 Ann Arbor—overwhelmingly, Comcast and AT&T—are not located in Ann Arbor, however. In fact, for a subset of locations around Ann Arbor we performed a `traceroute` to the reference server, and in all cases the shortest path from the wireless AP to the EECS network was through Chicago, Illinois. In some cases, in fact, packets went from the wireless AP in Ann Arbor to Chicago, to New York City, back to Chicago, then to the EECS network in Ann Arbor, only a few kilometers away. 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 τ 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 τ 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 τ. 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 Evaluation

In evaluating BreadCrumbs, we sought answers to the following questions:

- How accurately does BreadCrumbs forecast AP quality?
- How beneficial are such forecasts for applications that are commonly found on mobile devices?
- Is the overhead BreadCrumbs imposes reasonable for resource-constrained devices?

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

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 [16]). One of the authors carried the handheld with him continuously for two weeks during daytime hours (before seven pm). All data points lie in Ann Arbor, Michigan—population 114,000, density 1630/km^2 [8].

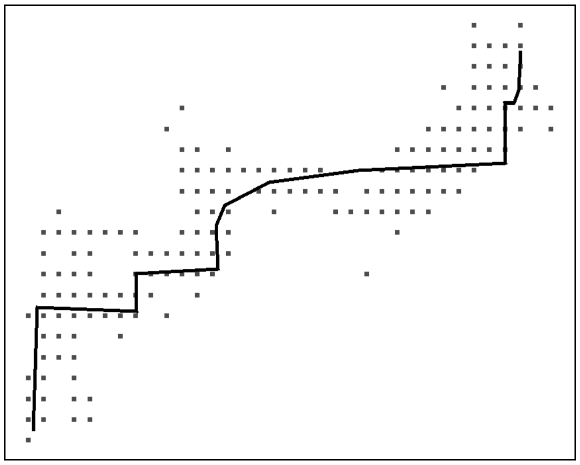


Figure 4: **Visited grid locations and *commute* ground truth**. Small squares are all GPS grid locations fixes from two weeks of user mobility traces collected. The black line is the “ground truth” path through the map taken by the user on his daily commute between home and work.

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 Ann Arbor’s latitude (approximately 42.2°N), 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

	mean	σ	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.

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.

Our prior work [25] showed that the application-visible quality of publicly-available access points varies significantly. The results in Table 2 support this conclusion. 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, 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 Bread-

	mean	σ	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.

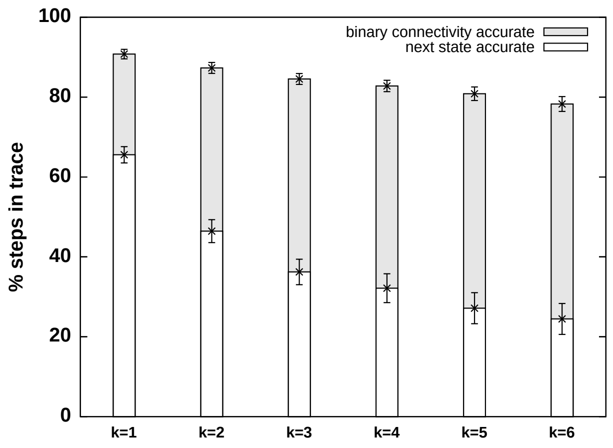


Figure 5: **Mobility model prediction accuracy**. k indicates the number of steps into the future BreadCrumbs forecasts.

Crumbs' 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 difference 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%.

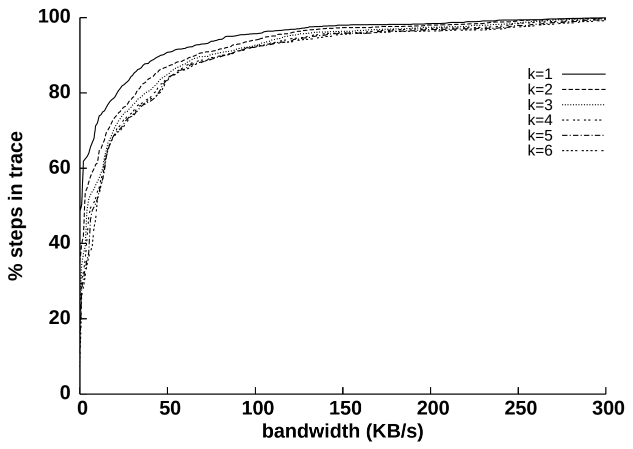


Figure 6: **CDF: bandwidth prediction error**. *k* indicates the number of steps into the future BreadCrumbs forecasts.

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 two algorithms that accomplished the same objective—one that was ignorant of any future predictions, and another that utilized BreadCrumbs' connectivity forecasts. For each trace in the evaluation set, we ran both algorithms, recorded the results, and subsequently averaged across all the runs.

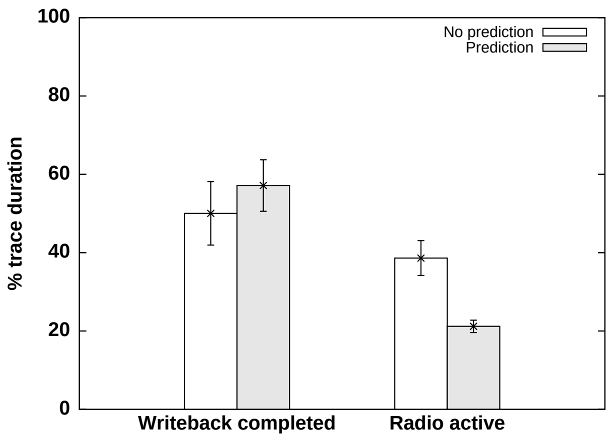


Figure 7: **Opportunistic writeback**. By utilizing BreadCrumbs' connectivity forecasts, the prediction-aware algorithm delays data writeback briefly to selectively use high-bandwidth access points. As a result, the total time until data was safe on the remote server is comparable, but the wireless radio is active 45% less often. This translates into significant energy savings.

5.3.1 Opportunistic Writeback

Our first 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 prediction-unaware 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.

We ran both algorithms once each 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 7 illustrates the results. On average, the prediction-aware 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 45% 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 total energy expenditure. For example, Anand et al [4] found that, for a 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.

5.3.2 Radio Deactivation

The previous section illustrated how a user-level application could employ BreadCrumbs' connectivity forecasts. Now consider how the operating system itself might make use of such information. A simple case is deciding when to deactivate a WiFi network interface in order to save power. The optimal policy would deactivate the interface whenever no usable access points are available, and activate it otherwise.

Determining if any usable APs exist at a given location requires power, however. Scanning for AP beacons, and possibly probing the bandwidth available via an AP, are all expensive operations. If the operating system activated the radio every few seconds, scanned for APs, and then made a decision, it may consume as much energy as simply leaving the radio active in power-save mode the entire time. Ideally, the operating system could query an oracle that would tell it what the current network conditions are, without having to turn on the wireless radio.

We devised two algorithms that sought to approximate this unattainable ideal–one that used BreadCrumbs' connectivity forecasts, and the other remaining prediction-unaware. As a reminder, each step the evaluation traces corresponds to ten seconds of real time (because BreadCrumbs scans for APs and calculates its GPS fix at that period).

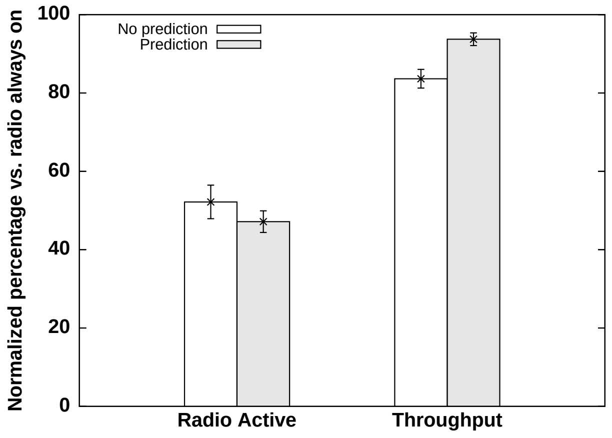


Figure 8: **Radio Deactivation**. All values are normalized to those of a baseline algorithm that keeps the WiFi interface active throughout the entire trace. The prediction-aware algorithm uses BreadCrumbs' connectivity forecasts to transfer more data while activating the WiFi interface less often than the prediction-unaware algorithm.

The prediction-unaware algorithm tracked the downstream bandwidth available to the device over the past 30 seconds in a sliding window. If at any point it detected that the device was disconnected for all of that time, it deactivates the wireless radio for the next 60 seconds. At the end of that period, the radio reactivates and the sliding window begins anew.

The algorithm that utilizes BreadCrumbs' connectivity forecasts works as follows. At each step of trace playback:

1. Query BreadCrumbs to get a connectivity forecast of the expected downstream bandwidth at the next three steps (10, 20, and 30 seconds in the future).
2. If the radio is on, and BreadCrumbs predicts no connectivity in the next 30 seconds, then turn the radio off.
3. Else if the radio is off, turn it on if BreadCrumbs predicts some connectivity in the next 30 seconds.

For the prediction-aware algorithm, when the radio is deactivated it obviously cannot calculate its GPS fix (because we are not assuming that a separate GPS radio exists on the device). Instead, when the radio is off this algorithm tracks real time and fires a state transition in the Markov chain every 10 seconds—choosing the state with the highest probability in the transition matrix. The next time the algorithm reactivates the WiFi radio, it can calculate its GPS location and "get back on track" if it strayed too far from the physical reality.

We also ran a third algorithm, which was simply a baseline that kept the radio active for the entire duration of all evaluation traces. All of the results for the other two algorithms are normalized percentages of this baseline. Our evaluation metrics were (1) total trace time the radio was active, and (2) data throughput from a remote host if downloading at full speed whenever the radio was active. Figure 8 illustrates the results.

The algorithm using BreadCrumbs' connectivity forecasts activates the WiFi interface for 47% of the trace, as opposed to 52% for the prediction-unaware algorithm. Despite this, the prediction-aware algorithm downloaded over 10% more data, 94% as much as the baseline algorithm that kept its radio active for the entire trace period. Connectivity forecasts let the prediction-aware algorithm only bother to "come up for air" when it is most likely to encounter usable APs, instead of just checking back periodically like the prediction-unaware algorithm.

5.3.3 Phone data network vs. WiFi

So-called *smartphones*, such as the Apple iPhone or the Nokia N95, are increasingly replacing traditional PDAs as users' mobile computing platform of choice. These devices can connect to the Internet through either a WiFi interface or over the mobile phone data network through a GPRS, EDGE or 3G connection. Each technology has positive and negative aspects. WiFi connections typically enjoy high bandwidth to the Internet, and are usually free of charge. Their range is limited, however, and coverage is nowhere near universal. Phone data networks, on the other hand, offer seamless coverage in most areas, but have much slower bandwidth and possible service charges. While 3G phone standards promise throughput rivaling that of WiFi, such services have not yet been widely deployed.

Even if 3G is available, however, one must still consider the power required to communicate with a distant phone tower, as compared to a lower-power connection to a nearby WiFi AP. In fact, Armstrong [5] found that WiFi was more energy-efficient than GSM data networks for all but the smallest transfers—those on the order of 30 KB. In a separate study, Agarwal [1] measured power consumption of both WiFi and EDGE during VoIP phone calls. Their results showed the WiFi interface consumed less than 75% as much energy as the EDGE link, for the same low-bandwidth (32 Kbps) call traffic.

For this portion of the evaluation, we consider an arbitrary application that is downloading data from a remote server. For example, this could be a file sharing client that is fetching files in the background, and can switch back and forth between using the WiFi interface and using the phone network. The current theoretical maximum throughput of EDGE is 384 Kbps (48 KB/s). Performance in real-world conditions is rarely this robust, however. Measurements of deployed network environments have found average EDGE bandwidth to be around 100 Kbps [15]. Even service providers themselves advertise these lower average rates. For example, AT&T (network provider for the iPhone in the United States) advertises average download speed on its EDGE network as 80-140 Kbps [6]. We chose a value near the middle of this range, 120 Kbps (15 KB/s), as the EDGE download bandwidth for the evaluation.

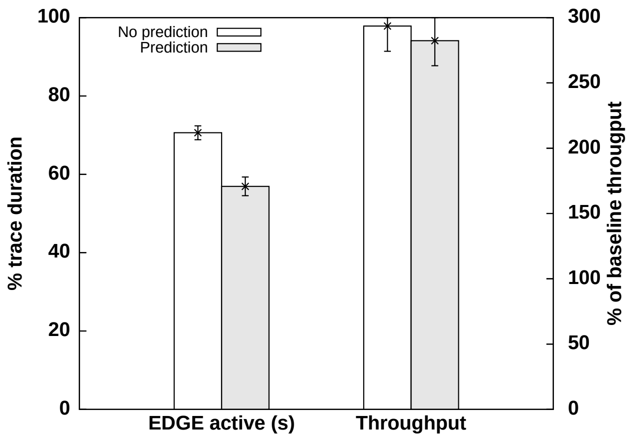


Figure 9: **Phone data network vs. WiFi**. The algorithm using BreadCrumbs' connectivity forecasts relies on the EDGE link 14% less often than the prediction unaware algorithm, but both achieve similar data throughput across the total trace duration. WiFi is preferred to EDGE when possible to conserve both energy and money.

We designed two algorithms to represent this application's behavior. They both attempt to use WiFi instead of the phone network as much as possible, while maximizing data throughput. The first is a prediction-unaware algorithm that simply uses the WiFi interface whenever there exists a WiFi AP whose downstream bandwidth is greater than the EDGE download bandwidth.

The second is a prediction-aware algorithm, which works as follows:

1. If there exists a WiFi AP at the current location whose downstream bandwidth is greater than the EDGE bandwidth, use WiFi at this location.

2. Else, query BreadCrumbs for its forecast of downstream bandwidth via WiFi during the next 30 seconds. Only switch to EDGE if BreadCrumbs forecasts there will be no WiFi bandwidth greater than the EDGE bandwidth during that interval. Otherwise continue using WiFi.

Briefly, the goal of the prediction-aware algorithm is to only resort to the EDGE network if the user is entering a true WiFi dead zone. One must consider the overhead required for handoff and association between WiFi networks, since

# 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.

such associations are short-lived in comparison to GSM networks. If the mobile device could associate simultaneously with multiple WiFi APs using one radio, BreadCrumbs could associate with and probe the quality of upcoming APs in the background while still connected to the AP currently used for data transmission. Such a capability is provided by VirtualWifi [10] for Windows-based devices, as well as by a virtual link layer we have developed for Linux.

We compare the results of these two algorithms with a policy that simply uses the EDGE link exclusively. Our evaluation metrics were (1) total trace time the application used the EDGE connection to transfer data, and (2) total data throughput. Figure 9 illustrates the results. Note the two y-axes in the figure. At left, we compare the percentage of total trace time the EDGE link was utilized. The baseline algorithm utilized the EDGE link 100% of the time, so both algorithms' results are a normalized percentage of this time. At the right, we compare the total download throughput of our sample application if transferring data continuously. This is scaled from 0 to 300% because the baseline algorithm that used EDGE exclusively transferred about one-third as much data as the two algorithms that were allowed to use WiFi when available.

The results are somewhat mixed. The algorithm that utilized BreadCrumbs' connectivity forecasts used the EDGE link 57% of the time, whereas the prediction-unaware algorithm did for 71% of total trace time on average. Since the WiFi link is more energy efficient in terms of joules-per-byte—particularly for large transfers such as this—this result is good news for device energy consumption. The throughput results, however, show the prediction-aware algorithm transferred 4% less data than the prediction-ignorant algorithm. Note that these values represent a theoretical cap on the throughput the device could achieve if actively downloading during the entire trace period.

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,

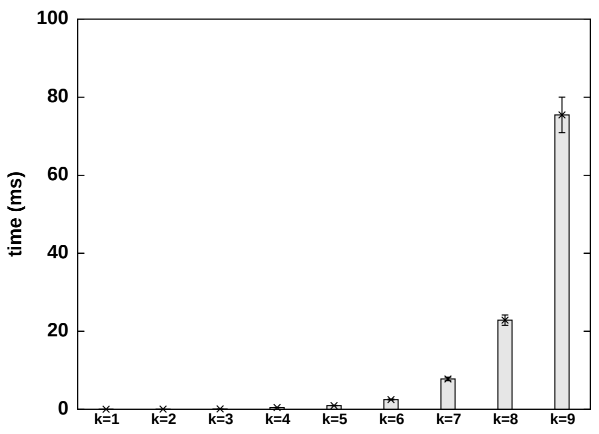


Figure 10: **Connectivity forecast overhead**. Results on a Compaq iPAQ handheld (400 MHz CPU), 128 MB RAM.

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 made did not implement caching of calculated forecasts or other possible optimizations in our implementation.

6 Related Work

MobiSteer [24] 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 [9], 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 our earlier project Virgil [25], optimize future resource discovery by caching historical access point quality information.

Song et al [28] studied the efficacy of applying different mobility prediction methods to the problem of improving bandwidth provisioning and handoff for VoIP telephony. Much like our work, they use real client traces to evaluate the success of a concrete application that is prediction-aware. They assume the existence of a centralized authority, however, that collects all mobility information, makes predictions, and disseminates instructions to the various wireless access points. We are focused on applications that are still useful when the device itself keeps its mobility history, and this information need not be disclosed to any other party.

Ghosh et al [13] 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.

Our prior work [31] concentrated, as did Kim et al [18], 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 [30]. 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 [23] 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 [17] 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 [7] use a variant of the LZ predictor described above to predict the next cell users will associate with. Yu and Leung [32] extend this idea to predict not only where a mobile device will hand off but also when this will occur. Liang and Haas [21] use a Gauss-Markov model in a similar way. Others use Robust Extended Kalman Filtering (REKF) [26], integrate individual path information with system-wide aggregate data [2], or estimate future locations through trajectory analysis [3]. Liu et al [22] 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 [20]. The same idea has recently been extended to use GSM phone towers rather than WiFi APs [11]. Fox et al [12] 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 [27] or mapping existing WiFi beacon sources [14].

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 two applications saw both improved performance and energy efficiency, while the results were somewhat mixed for the third.

References

[1] Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl, Kevin Chin, and Rajesh Gupta. Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Juan, Puerto Rico, June 2007.

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

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

[4] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Proceedings of the Ninth International Conference on Mobile Computing and Networking (MobiCom)*, pages 176–189, San Diego, California, USA, September 2003.

[5] Trevor Armstrong, Olivier Trescases, Cristiana Amza, and Eyal de Lara. Efficient and transparent dynamic content updates for mobile clients. In *Proceedings of the Fourth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 56–68, Uppsala, Sweden, June 2006.

[6] AT&T EDGE network, average throughput and latency. `https://cingular.atgnow.com/cng/tutorials/kb31719.html`.

[7] Amiya Bhattacharya and Sajal K. Das. Lezi-update: an information-theoretic approach to track mobile users in PCS networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 1–12, New York, NY, USA, 1999. ACM Press.

[8] United States Census Bureau. 2000 census of population and housing, summary population and housing characteristics, Washington, DC, USA, 2002.

[9] V. Bychkovsky, B. Hull, A.K. Miu, H. Balakrishnan, and S. Madden. A measurement study of vehicular internet access using in situ Wi-Fi networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2006.

[10] R. Chandra, P. Bahl, and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 882–893, Hong Kong, China, March 2004.

[11] Mike Y. Chen, Tim Sohn, Dmitri Chmelev, Dirk Haehnel, Jeffrey Hightower, Jeff Hughes, Anthony LaMarca, Fred Potter, Ian Smith, and Alex Varshavsky. Practical metropolitan-scale positioning for GSM phones. In *Proceedings of the Eighth International Conference on Ubiquitous Computing (UbiComp)*, pages 225–242, Irvine, California, USA, September 2006.

[12] Dieter Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July–September 2003.

[13] Joy Ghosh, Matthew J. Beal, Hung Q. Ngo, and Chunming Qiao. On profiling mobility and predicting locations of campus-wide wireless network users. In *REALMAN '06: Proceedings of the Second International ACM/SIGMOBILE Workshop on Multi-hop Ad Hoc Networks (MobiHoc)*, pages 55–62, Florence, Italy, May 2006.

[14] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom)*, pages 70–84, Philadelphia, Pennsylvania, USA, 2004.

[15] Timo Halonen, Javier Romero, and Juan Melero. *GSM, GPRS and EDGE performance: evolution towards 3G/UMTS*. J. Wiley, 2003.

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

[17] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, pages 244–251, Philadelphia, Pennsylvania, August 2005.

[18] Minkyong Kim, David Kotz, and Songkuk Kim. Extracting a mobility model from real user traces. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Barcelona, Spain, April 2006.

[19] David Kotz, Tristan Henderson, and Ilya Abyzov. CRAWDAD trace set dartmouth/campus/movement (v. 2005-03-08), March 2005.

[20] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place Lab: Device positioning using radio beacons in the wild. In *Procedings of the Third International Conference on Pervasive Computing*, pages 116–133, Munich, Germany, May 2005.

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

[22] 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.

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

[24] Vishnu Navda, Anand Prabhu Subramanian, Kannan Dhanasekaran, Andreas Timm-Giel, and Samir R. Das. MobiSteer: Using steerable beam directional antenna for vehicular network access. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Juan, Puerto Rico, June 2007.

[25] Anthony J. Nicholson, Yatin Chawathe, Mike Y. Chen, Brian D. Noble, and David Wetherall. Improved access point selection. In *Proceedings of the Fourth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 233–245, Uppsala, Sweden, June 2006.

[26] Pubudu N. Pathirana, Andrey V. Savkin, and Sanjay Jha. Mobility modelling and trajectory prediction for cellular networks with mobile base stations. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 213–221, New York, NY, USA, 2003. ACM Press.

[27] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking moving devices with the Cricket location system. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 190–202, Boston, MA, USA, 2004.

[28] Libo Song, Udayan Deshpande, Ulas C. Kozat, David Kotz, and Ravi Jain. Predictability of WLAN mobility and its effects on bandwidth provisioning. In *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Barcelona, Spain, April 2006.

[29] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive Wi-Fi mobility data. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1414–1424, March 2004.

[30] Jungkeun Yoon, Mingyan Liu, and Brian Noble. Random waypoint considered harmful. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1312–1321, March 2003.

[31] Jungkeun Yoon, Brian D. Noble, Mingyan Liu, and Minkyong Kim. Building realistic mobility models from coarse-grained traces. In *Proceedings of the Fourth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 177–190, Uppsala, Sweden, June 2006.

[32] Fei Yu and Victor C.M. Leung. Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks. In *Proceedings of the 20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 518–526, Anchorage, Alaska, USA, April 2001.