Practical User-Trained 802.11 Localization

Andrew Barry, Benjamin Fisher, Mark Chang
Olin College
1000 Olin Way
Needham, MA 02492
{andy, benjamin.fisher}@students.olin.edu, mark.chang@olin.edu

ABSTRACT
We present an indoor wireless localization system that is capable of room-level localization based solely on 802.11 network signal strengths and user-supplied training data. Our system naturally gathers dense data in places that users frequent while ignoring unvisited areas. By utilizing users, we create a comprehensive localization system that requires little off-line training and no access to private locations.

We have operated the system for over a year with more than 200 users working on personal laptops. To encourage use, we have implemented a live map that shows user locations in real-time, allowing for quick and easy friend-finding and lost-laptop recovery abilities. Through the system's life we have collected over 8,700 training points and performed over 1,000,000 localizations. We find that the system can localize to within 10 meters in 94% of cases.

Author Keywords
Location measurement, crowdsourcing, localization, location representation, location-based services

ACM Classification Keywords
C.2.4 Distributed Systems: Mobile Computing.

INTRODUCTION

Computerized localization, the automatic determination of position, will augment existing applications and provide opportunities for new growth. One can easily imagine a phone, computer, or other device changing behavior based on location. A phone might disable its ringer when in a conference or classroom. Calendar reminders would only appear if a user was not in already the event's location. A laptop could automatically select the closest printer when printing. Finding a colleague would be as simple as looking up a phone number.

Localization abilities have spawned a number of companies including GPS navigation [3][14], asset tracking [17][2], and E911 systems [4]. The most common implementation of automatic localization is GPS. While GPS performs well in many instances, it cannot achieve good accuracy indoors. To provide indoor localization, researchers have examined the use of dedicated hardware including ultrasound, IR, and RF beacons. Most of these platforms provide good resolution, but often have high installation, maintenance, and usage costs. With the advent of 802.11 wireless networking, researchers have turned to utilizing wireless access points as fixed RF beacons. This method mitigates the high hardware and installation costs of earlier systems, but often requires a substantial amount of off-line training, or the collection signal strength samples in many locations. Here we describe a system that uses 802.11 access points to localize, but transfers the training burden to the system's users, providing a cheap, fast, accurate, and low maintenance method for automated indoor localization.

To create a user-trained localization system, we provide a backend, client frontend, and the necessary interfaces. We also develop an interface that current users' locations that drives a friend-finding application. The backend includes a localizing algorithm and database, while the frontend provides users with the ability confirm, correct, or add to the community's data. We describe the system's deployment with over 200 untrained users for more than one year.

We use the following terminology to describe our system: a wireless *fingerprint* denotes the strengths of surrounding access points at a given location. A *bind* is the act of associating a fingerprint with a location. An *update* is a location scan and localization calculation. Fingerprints are collected automatically, binds are performed by users, and updates are performed automatically every five minutes or by user request.

RELATED WORK

Location-aware computing is not new. Perhaps the best-known location-discovery platform is GPS which uses U.S. government satellites to compute latitude and longitude [6]. While the high costs associated with GPS systems have disappeared, a receiver can only obtain a location with a clear sky-view. Moreover, GPS experiences substantial drift and is often not accurate enough to obtain room-level localization. Another set of commonly available localization systems serve the FCC's E911 initiatives [4]. These systems focus on approximately 100 meter accuracy and thus, like GPS, are of limited utility in indoor, room-level environ-

Submitted for review to UbiComp 2009.

ments.

The first indoor location-aware systems, such as Active Badge and MIT's Cricket, succeed with specialized hardware [15][10]. Active Badge uses wearable transmitters and a network of sensors to gather location information and report it back to a server. The Cricket system uses a combination of RF and ultrasound to provide accurate and private location data. These systems avoid training, but instead require a substantial hardware installation phase. Both Active Badge and Cricket require location-bound hardware that necessitates prior access by trained personnel to each desired localization area. This installation and the associated time and hardware costs limit these systems' wide-scale use.

As 802.11 networks became common, researchers began utilizing existing hardware to compute location. Microsoft's RADAR and later Haeberlen *et al.* show success in using the signal strength of 802.11 nodes to determine fine-grained indoor location [1][5]. These and similar systems require specialized training to create a database of location–signal strength tuples [11][8]. Training demands a substantial upfront effort and physical access to all of the desired areas. Moreover, after some time, the training data needs to be refreshed to account for changes in the environment and access point locations.

To reduce the expense of training, Intel Research demonstrates an algorithm that can estimate location with only minimal data by expanding its known area with continued use [7]. While the self-mapping algorithm costs little to implement, it requires a significant period of time to gain acceptable localization accuracy and coverage. Moreover, multiple radio configurations complicate the implementation of a shared-training system. Teller *et al.* introduce a crowdsourcing method that allows users to train and correct the system [13]. Teller's work, conducted in parallel with our own, is similar to the system described here although on a smaller scale in time, space, and number of users. They studied 16 trained users limited to a single building with a specialized platform for only 20 days. Here we present a year-long deployment of a similar crowdsourcing method with over 200 untrained users spanning five buildings operating on personal laptops.

Skyhook Wireless has combined similar crowdsourced data with a substantial set of training data to improve their worldwide 802.11 localization system. They use this "tier 2" data in a similar manner to that presented here, although their algorithms are not published [12].

ARCHITECTURE

Overview

We use a client-server architecture to enable fast, accurate localization and provide a mechanism for feedback. To localize, clients perform an *update* in which they collect wireless 802.11 signal strength information (a *fingerprint*) to send to a server. The server computes the client's location and sends the estimate back to the client for optional user review. The server also updates the friend-finding interface with the client's new location in case another user wants to locate the first. When the client receives the location estimate, it offers the user an opportunity to confirm or correct it (Figure 2). If the user chooses to take this opportunity (*binding* a fingerprint), the client sends the new ground-truth data back to the server, which stores the record for use in all future localization computations.

Figure 1. Interface in minimized mode.

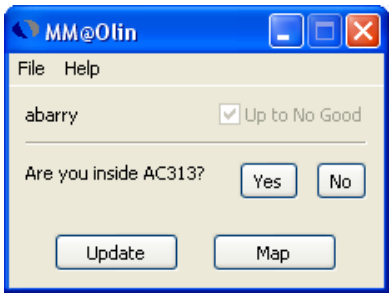


Figure 2. Typical interface. The client has localized and asks the user to confirm its estimate. The upper left displays a username and the upper right contains a humorous checkbox.

To implement our system, we require a sizable 802.11 wireless network deployment and users with some type of wireless device. While users must help train the system, their devices will not lose network connectivity due to our localizer. The signal strengths we utilize can be determined through "scanning" for networks without connecting. Thus, we do not affect the user's network connection and do not require the user to be able to connect to any access points, allowing us to leverage encrypted devices.

We now examine each part of the system in depth. We start with client's interface and proceed to discuss its implementation and communication protocol. We then discuss the server's implementation and localization algorithm. Finally, we examine the interface and implementation of our friend-finding service.

Client Interface

The user interface is designed to be as non-invasive as possible. Since the system is designed to perform on personal laptops, it must have a small resource footprint minimizing CPU, memory, and power consumption. We designed the client to autostart minimized in the system task tray (Figure 1) and never prompt the user without request. Given no user input, the client updates location every five minutes. If users want to train the system or access others' locations, double-clicking the icon brings up our deliberately simplistic interface. The GUI has two buttons: "Update" and "Map," which trigger a location update and show the campus-wide map, respectively. When an update is performed, the client displays its location estimation in question form, prompting a training response. The user can then accept the given

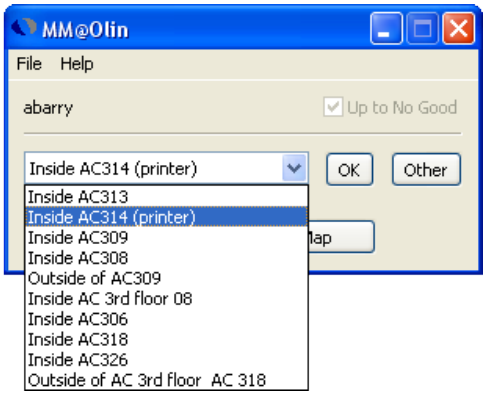


Figure 3. Training interface. The user has clicked "No" in Figure 2 and is now prompted with nearby locations. Clicking "Other" allows the user to create a new location point.

location, choose from a list of likely locations, or create a new point. A full training interaction takes 5 to 30 seconds, depending if a new point is created (approximately 5% of training instances include new point creation). Figures 2–4 show typical GUI screens.

Unlike MIT's method which uses blueprints to preload all known locations, our system allows users to create new places [13]. When the user determines that the localization is not correct, he or she can decide to create a new location. The client prompts for the building, floor, and a text name, suggesting a room number or descriptive phrase for the text entry (Figure 4). Once a user has entered those data, a labeled map of the campus appears allowing the user to select where the new location will appear visually. We found that making this process intuitive is key to ensuring the success of a crowdsourced data collection application.

New point creation is challenging for both the user and GUI designer. Most users are unable to correctly identify their location on an unlabeled blueprint, so the user interface must be copiously labeled to prevent errors. We chose to allow users to customize location names, making their descriptions much more useful to others. For example, instead of calling a room "335," users labeled it "3rd floor lounge."

Allowing for free-form input, however, provided substantial possibilities for error. Although "lounge" is a natural input, it is not useful without context. To obtain both flexibility and accuracy, we constrain building and floor choices while allowing for a textual description. Thus, points have reliable context information and custom labels.

Finally, we note that custom naming lowers the entry barrier for new localization systems. We need only building diagrams while other systems require blueprints or CAD models to generate locations. With time, users supply all the necessary data.

Client Implementation

We implemented the client using wxPython, a cross-platform GUI toolkit that allows us to deploy the same code-base

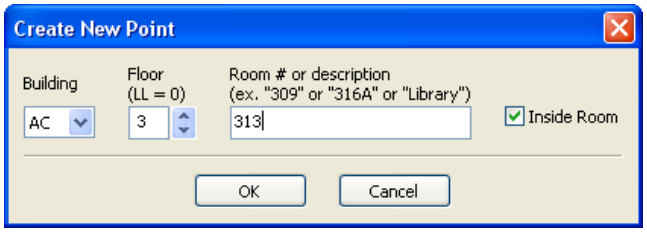


Figure 4. New location creation interface. The user has clicked "Other" in Figure 3 and is now prompted to enter the details of his or her location. After clicking "OK," the user will be prompted to select the location on a map of the local area.

across many common operating systems [18]. We concentrated on making the interaction simple and intuitive so users would not require training. For example, instead of asking the user for his or her name, we simply send the client's system-username to the server, which translates it into a full name. With almost all users using the standard "jsmith" for "John Smith" format, this translation was straightforward.

Wireless Fingerprint Collection

Obtaining the wireless signal strength data on a variety of platforms was a challenge. Almost all wireless information is accessible on both Mac and Linux platforms through standard system calls. On Windows XP, however, we were not able to successfully capture the data through a system call. Luckily, the Intel ProSet/Wireless driver places these data in Windows Management Instrumentation (WMI) which we can access. Since almost all users were running that driver, this solution was acceptable. We do not support other Windows-based platforms, although adding such support is trivial now that better methods for obtaining signal strength data are available [9].

We use computer-to-access-point signal strength, reported in dBm, for localization. To create a fingerprint, we concatenate a series of wireless access point MAC addresses and signal strengths. MAC addresses are unique, even when two access points are configured to broadcast the same network name, so we use them to index signal strength. Once we have compiled a fingerprint, we make a simple HTTP GET request, the same request used to load most webpages, and append the fingerprint and username to the URL. The server's PHP/MySQL engine then computes the estimated location and produces an HTML page that the client interprets and displays. Thus, our entire communication process consists of a single webpage load.

Backend Services

When localizing, a client queries the server with a wireless fingerprint and identifying information appended to the URL. Our backend, built on a standard LAMP (Linux, Apache, PHP, and MySQL) stack, reads those variables and produces a location estimate. When training, the client performs a similar HTML page-load, but this time to a specialized training PHP script. The training script reads the client's data in a similar manner and appends the new fingerprint to the location records for use in a later localization. Here we discuss

the implementation of both of these components.

Position Estimation

We compute locations using a Euclidean distance algorithm, similar to RADAR's Nearest Neighbor in Signal Space (NNSS) algorithm [1]. When a fingerprint is received, the backend localization process produces a SQL query that substitutes the fingerprint's signal strengths into a Euclidean distance equation. We then iterate through the database, finding the signal-space-distance between our fingerprint and each known location, and choose the location that minimizes distance. Equation 1 shows a single iteration of this sequence:

$$D(N) = \sqrt{(C[1] - F[1])^2 + (C[2] - F[2])^2 + ... + (C[76] - F[76])^2} \quad (1)$$

N denotes the current location candidate and ranges from 1 to the number of known locations, which grows with each training event, as explained below. C denotes the array of location N's stored fingerprint values. F denotes the array of the client's fingerprint values. We choose the N that minimizes D as the location estimation. In our implementation, each fingerprint array contains 76 values, the number of access points on campus.

By minimizing D, we find the fingerprint that is closest to the user's scan in a 76-dimensional Euclidean space. Implementing this computation in SQL allows us to perform the estimation rapidly over a large set of possible locations. Listing 1 shows the algorithm in SQL form. Note that we do not use a square-root in the SQL implementation because it is unnecessary for the minimization function.

```
SELECT placename, min(pow(C1 − F1, 2) + pow(C2 −
    F2, 2) + ... + pow(C76 − F76, 2) AS score FROM
    point WHERE 1 GROUP BY placename ORDER BY
    score ASC LIMIT 10
```

Listing 1. SQL code for localization. *C1* represents the recorded signal strength from the first access point for the candidate location. *F1* represents the signal strength of the first access point in the client's fingerprint.

Training

The client-server interaction for a training event is very similar to the localization transaction. After a localization, the user enters ground-truth data which the client sends to the server along with the location's wireless fingerprint. The server reads those data and appends them to the known-location database. All points, new locations or confirmations, are handled identically. In each case, the script adds the new fingerprint without regard for the database's content. While this duplicates information such as location name, it greatly simplifies training and has no adverse impact on localization performance. An example illumatates the process: given two locations, *A* and *B*, two pieces of user training data for *A*, and one piece for *B*, the database would look like Table 1.

Friend-Finding Service

Figure 5. Search interface. Without auto-completion, users had difficulty finding their friends.

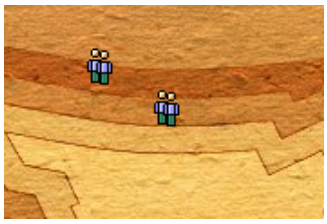


Figure 6. Floor interface. Darker edges indicate a lower floor. In this case, two people are located on the ground floor and two more are located on the first floor.

Figure 7. Custom icons allow quick and easy user identification. The user in the upper right is faded to indicate that he or she has not reported location recently.

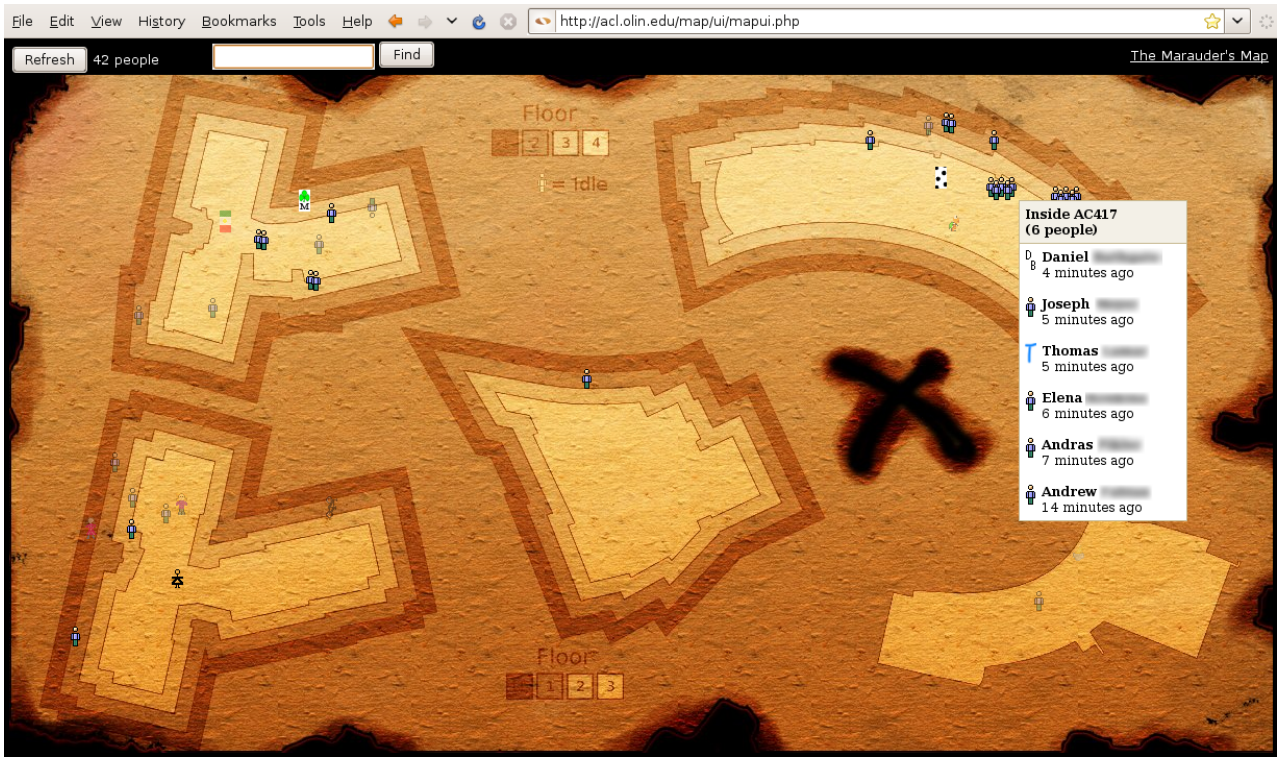


Figure 8. Friend-finding webpage frontend. The interface is themed like an old magical map. The user has moved the mouse over a cluster of people, prompting a drop-down list of location, names, and update times.

To motivate users to train the system, we implemented a friend-finding service that publishes user locations on a map-like interface. The goals of the service are threefold: allow users to quickly and easily locate specific people, display all users' locations on one screen, and act as an advertisement for the project. Here we describe the service's frontend interface and backend logic.

To satisfy our goals, the interface must allow for both searching and browsing while maintaining an attractive look and feel. To encourage use and help explain the concept, we themed the project as the "Marauder's Map at Olin," a reference to a magic map that displays people's locations in the popular Harry Potter series (Figure 8). While this theming might seem trivial, it is critical to building and maintaining a user base. The map theme helped people understand why the service is useful and encouraged them to share with their friends. We implemented searching through a simple text-input box, although an auto-complete mechanism was necessary to help users spell names correctly (Figure 5).

Location	Access Point Signal Strengths (dBm)							
A	50	30	27	45	...	...	...	11
B	24	29	24	41	...	...	...	47
A	45	39	22	39	...	...	...	7

Table 1. Sample database rows for locations *A* and *B*. *A* has been trained twice and *B* only once. Note that we add 100 to the dBm reading for convenience.

Displaying hundreds of users on multiple floors proved to be a challenge. Early tests showed that users clustered near the edges of buildings, so we expanded our building representations to show the edges repeatedly, utilizing the new space to indicate vertical displacement and avoid icon overlap (Figure 6).

We implemented small customizable icons to allow users to identify friends at a glance. The client interface supports the ability to upload a 11x21 pixel icon that is used to display location. With practice, this allows instant identification of most users. When icon identification is insufficient, we display the name, time since update, and location on mouse-over. We also fade user icons to show inactivity when clients have not updated in 20 minutes. After an hour without updates, we remove the user's icon from the map, although one can still find the user's last location by searching. This longer-term use is for asset localization, such as a lost-laptop scenario.

We implemented this interface using a combination of PHP, MySQL, and JavaScript. When the user accesses the friend-finding page, a PHP script queries the current-location database which returns the locations of all users. A JavaScript engine then overlays the appropriate icons on the map background and creates the mouse-over text fields. With a relatively small user base, we are able to download all users for local searching. If implemented on a substantially larger scale, we would need to consider creating an AJAX-enabled search tool.

Privacy

Privacy is a concern in any localization system. Given that the primary application of our implementation is a friend-finding service, we found that concerned users were aware of the implications and simply chose not to participate. Some users requested the ability to remove their location report at any time, a feature we implemented. All of the system's services reside only on internal servers to ensure that location information is not published outside of our institution.

Initial Training

Our system's novelty stems from the method used to gather wireless fingerprint data. Similar systems require a substantial training effort before coming online. Typically, this type of training takes about 1-3 minutes per location [5]. At this rate, a system of our size would take approximately 16 person-hours to train. Instead, we train the system to a minimally usable state, taking less than 20 seconds per location for 300 scans, or about 1.5 hours of training. With that training, we create a sparse map of locations, allowing for reasonable (within 10-20 meters) estimates. This initial training convinces users that the system is useful enough to train themselves.

APPROACH TO CROWDSOURCING

Motivation

The primary motivation for crowdsourced data is the reduction in time required to train the localizer. Moreover, with crowdsourcing, users provide most of the data while the system is already localizing, reducing the time before the localizer can be used. When training, researchers found that gaining access to private and semi-private spaces (offices, dorm rooms, etc.) was difficult and awkward, a problem that user-trained systems avoid [5].

A pre-trained system requires retraining to account for changes in the environment, but a user-trained system is continuously updated with no overhead. In addition, crowdsourced training naturally produces data that are dense in places that are commonly visited. Users tend to bind in places they frequent, causing common locations to have dense data. Because our localizer treats each bind separately, it weights common locations more heavily, resulting in a natural location-frequency dependency.

Finally, traditionally trained systems suffer from a conflict between coverage, or the number of distinct locations with data, and accuracy, or the measure of how often the localizer chooses the correct location. If the system is aware of many often unoccupied locations, it will suffer from a decrease in accuracy. Crowdsourcing helps mitigate the problem by naturally ignoring rooms that users do not frequent. For example, there are small, narrow trash rooms in each wing that were never bound in our system (Figure 9). A traditionally trained system might incorrectly place a user in one of these rooms, even though the probability of a user being located there is very small. Our system will always avoid these areas because no users have bothered to train them.

UI Considerations

The UI of a crowdsourced application is paramount to its success. To retain users and develop a large training base, we ensure that our UI is simple, intuitive, and non-intrusive. College students with only standard computer experience must be able to understand, use, and train the system with no instruction. To avoid annoyance, the application does not notify the user in any way, without a deliberate interaction.

We chose to provide our primary service, friend-finding, without requiring the user to run the localization client. We made the application easily available from within the client, but not dependent on it. This tactic was successful, as we found that the service provided effective advertising for the client.

RESULTS

After a short beta test, we deployed our system campus-wide at Olin College in April 2008 and have continued operations for over a year. Olin has approximately 300 students in five buildings, encompassing more than 300,000 square feet. We announced the project with an email to a common list in which we explained the concept and encouraged users to download and run the client. To date, we have had more than 200 unique users, 8,700 binds (95% of users contributing), and over 1,000,000 location updates.

One advantage that our environment provides is that a large majority of students own Dell Latitude D-series laptops. Although we support Windows, OS X, and Linux, most of our users use the same driver and wireless chipset.

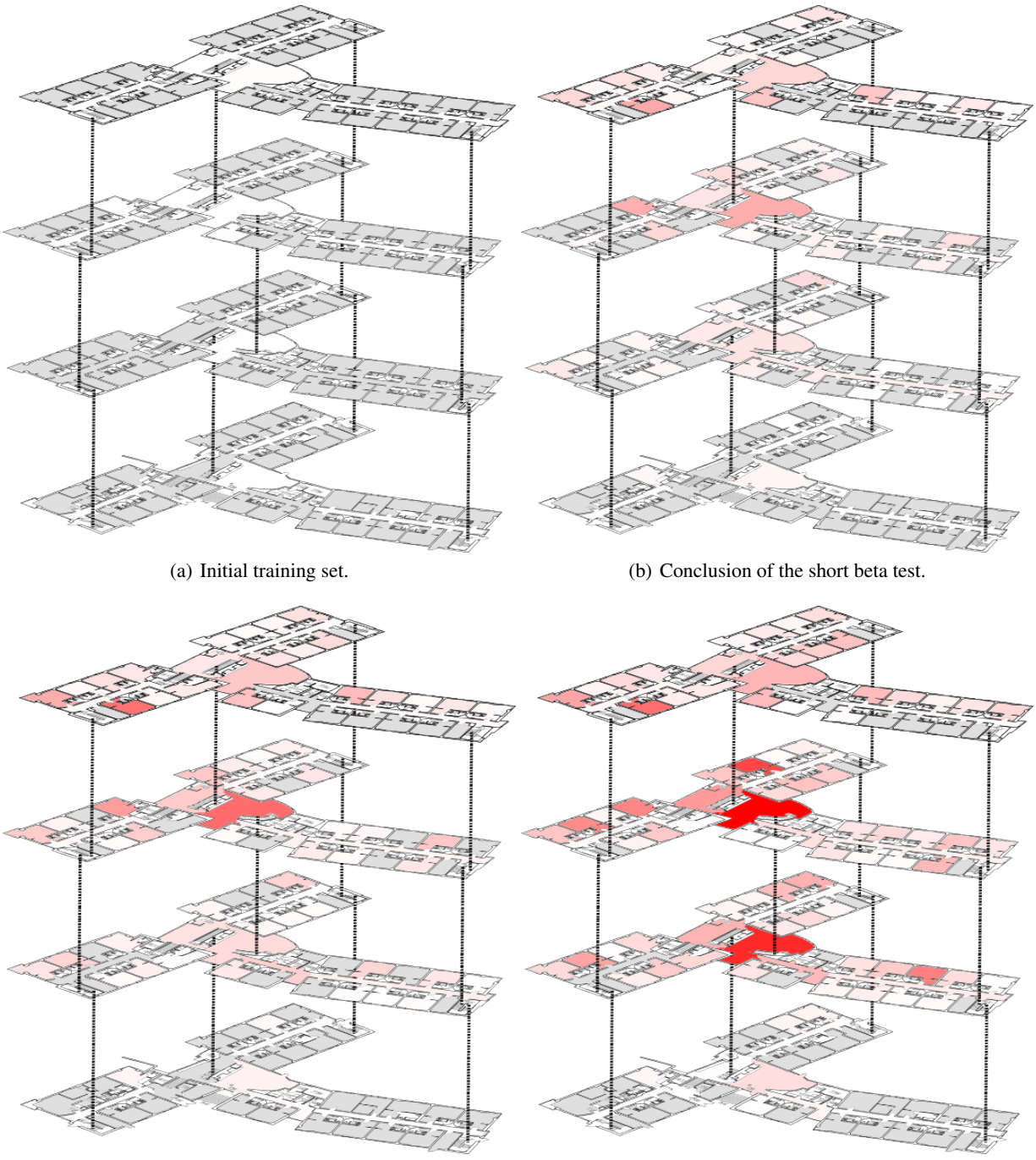
System Accuracy

A localization system's accuracy is determined by both *coverage* and *accuracy*. Coverage measures how much of the deployment area has associated fingerprints. Accuracy is the measure of how often the localizer reports the correct location. We first discuss coverage and then proceed to examine our localizer's accuracy.

At the beginning of deployment, only our initial survey supplied data, so we started with poor coverage, especially in private rooms (Figure 9(a)). We found that within 30 days of launch, our coverage stabilized at a reasonably complete level (Figure 9(c)). Coverage progression for one building can be see in Figure 9. Other buildings show similar patterns, although places students are unlikely to spend time, such as faculty offices, never achieve good coverage.

Our second metric is system accuracy. Accuracy starts poor and improves with the number of binds. After the system had reached a steady state, we emailed our users a survey asking them to report the localizer's accuracy at that moment. This method has the advantage of testing in locations that users frequent, which represents our actual accuracy better than a random survey of locations would.

Figure 10 shows localization errors. We find that we localize to within 5 meters in 69.9% of attempts and to within 10 meters in 94.9% of cases. This accuracy is approximately equal to other published systems, although it does not achieve the accuracy of Haeberlen *et al.*'s calibrated technique [5][13].



(a) Initial training set.

(b) Conclusion of the short beta test.

(c) 30 days after launch.

(d) One year after launch.

Figure 9. Map of training density in one of the five deployment buildings. Gray indicates no training data. Light to dark red indicates progressively more fingerprints for that space. Note that common areas (center of the building) have far more data than individual's rooms.

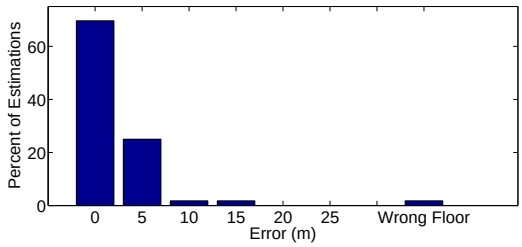


Figure 10. Localization error. We determined error by asking users to perform spot checks in their current location. We find that we localize correctly in 69.9% of attempts and are within 10 meters in 94.9% of attempts. We localize to to the wrong floor only 1.8% of the time.

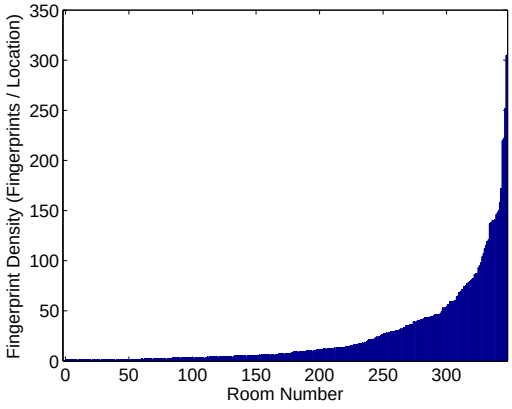


Figure 11. Fingerprint density per location. A number of common areas have substantially more signal strength data than most locations. As predicted, these areas are the most frequented by our users.

We note a number of conditions that degrade our performance. First, during our deployment, a large fraction of the campus's access points were converted from WEP to WPA2 encryption, resulting in a change in MAC address. The system does not recognize the new configurations and assumes that none of the old access points exist. In addition to changing MAC addresses, the administrators moved a smaller fraction of access points to improve wireless performance. While users have retrained the system, this movement continues to degrade our performance.

Supporting a wide range of laptops, including Mac hardware and four models of the Latitude D-series, also detracts from accuracy. We currently do not correct for radio/antenna configurations, although the system naturally has some correction based on who binds, as described below. Finally, we note that user-generated data are not as reliable as professionally generated scans.

Training Density

As expected, we collected more fingerprints in common and public locations than in private rooms. The median number of binds per room is 7 and the maximum is 305 which occurs in a residence hall lounge. 17% of known locations have only one bind and 53% have 10 or fewer. Figure 11 shows the number of binds per location.

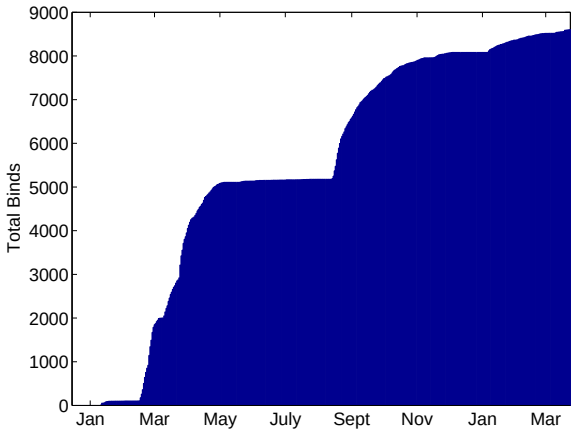


Figure 12. Fingerprint database size over time. Training rates were steady after release and have reduced as the system became more accurate. We are not surprised to see little data added during the summer term when students are not on campus.

User Behavior

After release, we collected about 71 binds per day and within 2 months our data set had grown to 27 times the initial training, a collective effort of approximately 25 person-hours in ideal conditions. While training rates decreased as accuracy increased, users still train the system today. Figure 12 shows these trends in database size over time.

We found that the user contribution profile is similar to other mass-interaction applications [16][13]. A few enthusiastic users bind an inordinate number of times. Interestingly, these users do not update their location as often as we might expect (Figure 13).

As the system's coverage grew, the number of new locations bound quickly decreased. Our initial survey bound 16% of total locations, beta testers bound 48%, and our general user base bound the remaining 36%. Thus, we find that users are far more likely to pick an existing location than they are to bind a new one.

We hypothesized that our data collection method would result in dense data in frequented places, and our expectations are confirmed. We see this in the similarity between Figures 14 and 9(d) which show where locations are reported and bound, respectively. These data confirm our second hypothesis, that users bind in places that they, individually, frequent. 51.2% of all location updates occur in places that users had bound themselves. In other words, half the time a user is in a place where he or she has contributed data. Thus, we confirm that one of the primary advantages of crowdsourced of data collection is that users reside where their own data are best.

Application Use

Our friend-finding service was key to obtaining data over the long term. The service convinced new users to join and provided an incentive for continued use. In one year we logged over 14,000 friend-finding page loads.

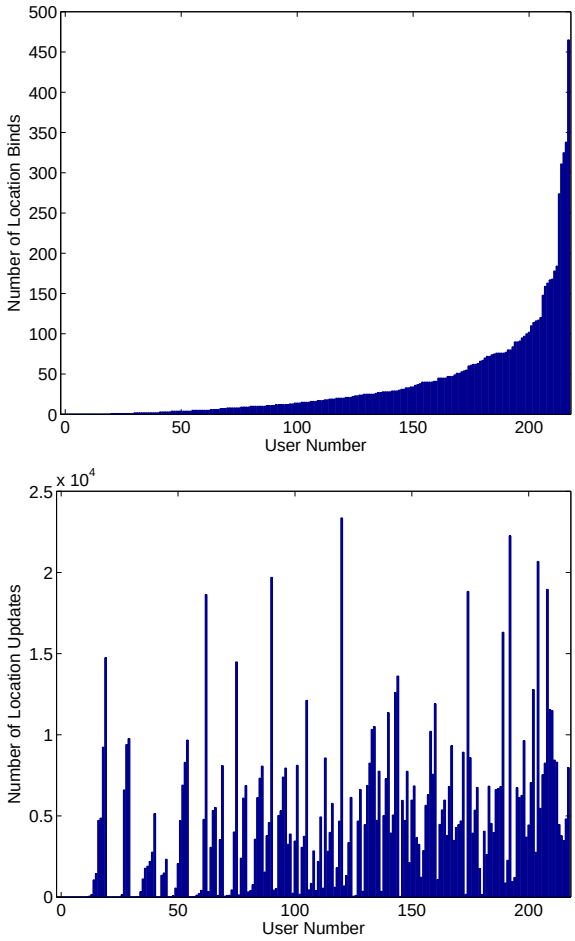


Figure 13. Number of binds by a given user and their respective number of location updates. We see that users who bind often are not the primary consumers of their data.

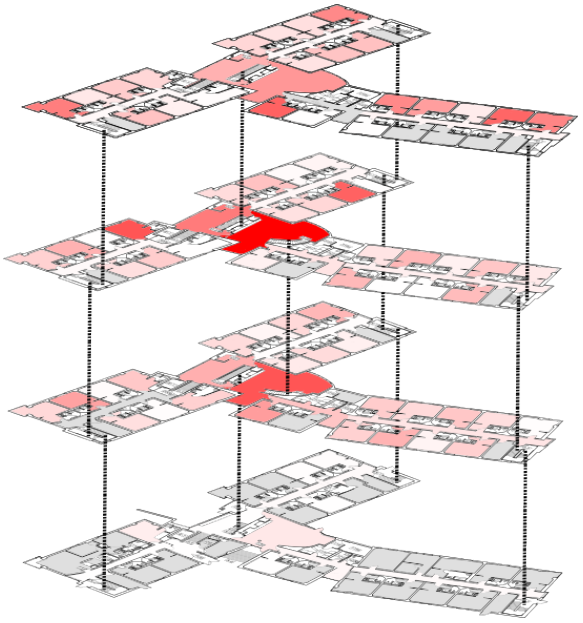


Figure 14. Combined density of location reports for one year in one residence hall. Clearly, common areas are visited far more often than individual rooms. No students live on the first floor so we are not surprised to see few reports in that location.

FUTURE WORK

This paper presents an implementation of a user-trained localization service covering an entire college campus. To utilize this framework in other scenarios and improve localization robustness, we consider a number of additions. First, we examine database architecture improvements, followed by new applications, and finally new ways to collect training fingerprints.

Architecture Improvements

To support larger and more diverse systems, we are considering redesigning our database architecture to support automatic addition of previously unknown access points. Moreover, a redesign would also allow us to implement searching over a limited set of fingerprints, decreasing query latency. Currently, latency is not an issue but substantially larger installations might begin to see significant slowdowns.

We saw degradation in performance when access points moved and MAC addresses changed. While new training fingerprints helped to mitigate the issue, old fingerprints continued to mislead the localizer. We look forward to implementing a weighting system that favors new data, marginalizing old and possibly outdated data. The design of the weighting system requires further study to determine if it should universally downgrade old data or only ignore old fingerprints when newer ones are available for a location. The first implementation would cause the localizer to favor newly bound points while ignoring old fingerprints, effectively reducing coverage over time. The second implementation retains coverage, but might not reflect newer user-movement patterns.

New Applications

To augment our user base, we are considering implementing additional localization-based services. Preliminary surveys have shown substantial support for an application that automatically selects the user's closest printer, preventing the common mistake of printing to another building. We are also considering a tagging system for files that records location information on creation and modification. With this information, a user could search for all files created in a particular room. Users tend to create different types of files in different places, such as minutes in a meeting room and source code in a lab, so searching based on location might be helpful.

We are also considering expanding the number of platforms we support. Porting our code to hand-held and smart-phone devices would provide interesting new data sources and challenges. A more diverse set of hardware platforms might require a platform identification mechanism and conversion functions between device fingerprints, like Haeberlen's implementation across different radio platforms [5].

Alternative Training Methods

To gather more binds, we are considering automatic calendar integration. Many calendar appointments are tagged with a location which we could extract and utilize to automatically train the system. When a user's calendar indicates that he or she is in a specific place, the system could automatically collect fingerprints and bind them to that location. Obviously, users and/or their wireless devices are not always located where their calendar indicates, but with intelligent use of idle-times and a limited fingerprint set, this type of training could be made reliable. In more extreme cases, we might consider using calendar integration to train an entire system without any user interaction, although the accuracy of these fingerprints would surely be questionable.

CONCLUSION

We have described a user-trained system that performs accurate indoor wireless localization in areas with existing 802.11 networks. The system can be deployed in any location that has a pervasive network and a group of users willing to train it. By utilizing personal laptops and existing access points, we do not need to build or buy any additional hardware. The system's interfaces are simple and intuitive, allowing users to localize, find others, and contribute training data with no instruction.

After more than one year, our system continues to operate and has accumulated more than 200 users, 8,700 binds, and over 1,000,000 location updates. Usage patterns provide natural guidance for the localizer, improving accuracy by accumulating dense data in common areas. These methods result in successful localization to within ten meters in over 94% of cases, providing convincing evidence that crowdsourcing is a practical method for cheap, pervasive wireless localization.

ACKNOWLEDGEMENTS

We would like to thank the community at Olin College for their ideas, testing, and feedback. Andrew Barry and Benjamin Fisher are supported by F. W. Olin Scholarships.

REFERENCES

1. P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proc. IEEE Infocom 2000*, pages 775–784. Proc. IEEE Infocom 2000, IEEE CS Press, 2000.
2. Ekahau, Inc. `http://www.ekahau.com`.
3. Garmin Ltd. `http://www.garmin.com`.
4. D. Geer. The E911 dilemma. *Wireless Business and Technology*, Nov. 2001.
5. Haeberlen, Flannery, et al. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of the Tenth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Sept. 2002.
6. J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, Aug. 2001.
7. A. LaMarca et al. Self-mapping in 802.11 location systems. In *UbiComp 2005: Ubiquitous Computing*, pages 87–104. LNCS 3660, Springer, 2005.
8. Letchner, Fox, and LaMarca. Large-scale localization from wireless signal strength. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2005.
9. MSDN Reference: WlanGetAvailableNetworkList Function. `http://msdn.microsoft.com/en-us/library/ms706749(VS.85).aspx`.
10. N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *6th Ann. Intl Conf. Mobile Computing and Networking (Mobi-com 00)*, pages 32–43. ACM Press, 2000.
11. Skyhook Wireless. `http://www.skyhookwireless.com`.
12. Skyhook Wireless Privacy Policy. `http://skyhookwireless.com/whoweare/privacypolicy.php`.
13. S. Teller et al. Organic indoor location discovery. Technical Report MIT-CSAIL-TR-2008-075, MIT, Dec. 2008.
14. TomTom NV. `http://www.tomtom.com`.
15. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 40(1):91–102, Jan. 1992.
16. S. Whittaker, L. G. Terveen, W. C. Hill, and L. Cherny. The dynamics of mass interaction. In *In Conference on Computer-Supported Cooperative Work (CSCW)*, Nov. 1998.
17. Wisetrack, brand of TVL, Inc. `http://www.wisetrack.com`.
18. wxPython. `http://www.wxpython.org`.