

A Probabilistic Room Location Service for Wireless Networked Environments

Paul Castro¹, Patrick Chiu², Ted Kremenek¹, Richard Muntz¹

¹UCLA, Department of Computer Science, Los Angeles, CA, 90095, USA
{castrop, kremenek, muntz}@cs.ucla.edu

²FX Palo Alto Laboratory, 3400 Hillview Ave, Bldg 4, Palo Alto, CA 94304, USA
chiu@pal.xerox.com

Abstract. The popularity of wireless networks has increased in recent years and is becoming a common addition to LANs. In this paper we investigate a novel use for a wireless network based on the IEEE 802.11 standard: inferring the location of a wireless client from signal quality measures. Similar work has been limited to prototype systems that rely on nearest-neighbor techniques to infer location. In this paper, we describe Nibble, a Wi-Fi location service that uses Bayesian networks to infer the location of a device. We explain the general theory behind the system and how to use the system, along with describing our experiences at a university campus building and at a research lab. We also discuss how probabilistic modeling can be applied to a diverse range of applications that use sensor data.

1. Introduction

One approach to ubiquitous computing is enabling applications and devices to detect and make use of changing environmental conditions. Such systems are also called *context-aware*, and important aspects of context include location, nearby people, and accessible resources (e.g. see [7], [15], [18]). In this paper, we address the problem of inferring the room location of a device such as a laptop or PDA in a wireless networked environment. Such environments are becoming more prevalent due to the popularity of wireless local area network products based on the IEEE 802.11 standard (Wi-Fi). These products offer good bandwidth and affordability for office buildings, university campuses, and homes.

Location-aware applications that have been explored or proposed include tour guides [1], interaction with or control of nearby computers, displays, and printers [18], and electronic Post-it notes associated with a location [3]. Like most existing location based systems, these require special hardware to detect the location of the device running the application. For example, one method to do location sensing is to employ an infrared transceiver system, as [18]. Another way to obtain location information is with commercially available GPS, which only works outdoors. By following the device until it enters a building, it can locate the nearest building in a research park or campus.

Providing a location service that requires no extra hardware would make it easier to build location-aware application systems and enable more people to use them. In a Wi-Fi environment, it is possible to use software to infer the location of a wireless networked device by analyzing the signal strength or signal-noise-ratio of the wireless access points with respect to that device. By using software to infer location, no additional hardware needs to be attached to a laptop or PDA beyond a wireless network PC card.

Some techniques that may be applied to inferring location are multilateration, nearest neighbor, and Bayesian networks. For example, in multilateration, an object can infer its location by calculating its range from beacons with known locations using some type of signal measure like radio frequency (RF) or ultrasound. This works fine in environments of uniform density, but the walls and other structural as well as non-structural matter inside a building makes calculating ranges from signal measures difficult. In a sense, the space is warped. A nearest neighbor approach that compares signal samples in a table of values can work, but lacks a well-defined model for combining data from multiple sources. Our choice is to apply Bayesian networks, which can “learn” the room locations of a building, and its modular structure provides flexibility when the access points are added/removed or spatially reconfigured. A further advantage is that other contextual information such as the likelihood that the owner of the device will inhabit a particular location can be easily incorporated into the Bayesian network model.

Another important consideration is that the wireless signals undulate, which is caused by people walking around and other changes in the ambient space. This “noise” is handled by a Bayesian network, which assigns probabilities to locations.

The UCLA Multimedia Systems Lab has devised and implemented the Multi-use Sensor Environment (MUSE), a middleware architecture for sensor smart spaces employing Bayesian networks [4]. A location service for Wi-Fi networked environments was prototyped on top of this architecture, and a stand-alone version of this location service called *Nibble* was built. *Nibble* has been installed for evaluation at Boelter Hall, a university campus building at UCLA, and more recently at an industrial research lab, FXPAL. In this paper, we will describe how this location service works (section 2), our experiences with this system (section 3), and we will discuss ways of extending the MUSE sensor model, which is inherently simple because of the structure of the Bayesian networks, to compose component-based location services (section 4). Finally, we discuss plans for future work (section 5).

2. Location Service System

The IEEE 802.11 Wi-Fi wireless network infrastructure consists of many wireless clients and several access points that act as bridges between the wireless network and the local LAN. Clients primarily use, or “associate,” with the access point that

provides them with the strongest RSSI (received signal strength). As a client roams, it periodically does a site survey of signal quality measures to determine the best access point with which to associate. As reported in [2] and [16], it is possible to infer the location of a client from these signal measures. Past approaches have been limited to nearest neighbor or error-minimization techniques to infer location from signal strengths. In this section we describe a modular probabilistic approach for inferring location that uses Bayesian networks. Through probabilistic modeling we can create a robust location service with a general, quantifiable measure of performance, easily incorporate additional data such as movement histories of users, as well as provide a flexible means to compose a location service out of heterogeneous components.

2.1 System Overview

Nibble is a stand-alone version of a Wi-Fi location service that is derived from an indoor location service prototyped within the MUSE pervasive sensor infrastructure [4]. Nibble relies on a *fusion service* to infer the location of a laptop from signal strength measures between itself and access points in the environment. A fusion service has two components: 1) an evidential reasoning model that aggregates and interprets (i.e. fuses) information from sensors and 2) a resource optimization model for minimizing the “cost” of gathering data. Sensor data is characterized probabilistically and inputted into the fusion service. The output of the fusion service is a probability distribution over a random variable that represents some context data. This process is necessary since sensor information, such as Wi-Fi signal quality measures, is fundamentally noisy, subject to intermittent availability, and typically too-low level to be immediately useful to an application.

Bayesian Network Framework for Inferring Location

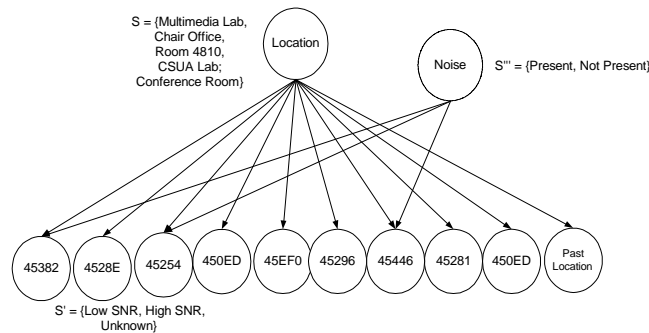


Fig. 1. Example Bayesian network for inferring location in a Wi-Fi network

In Nibble, the interpretive part of a fusion service is a Bayesian network (also called *belief network* or *causality network*). A Bayesian network is a graphical representation

of a joint probability distribution that explicitly declares dependency relationships between random variables in the distribution [12]. Because these relationships are declared, a joint probability distribution can be constructed modularly thus making computation on this distribution feasible. In our view, the modular structure of Bayesian networks also eases the construction of component-based systems where the outputs of one fusion service can be used as an input to another.

Figure 1 shows a simple Bayesian network we can use to infer location from the Wi-Fi infrastructure. The network is a rooted tree with directed arcs going from the root to several terminal nodes. These arcs signify a dependency relationship between the root node and the terminal nodes. The root node is the “query” variable that describes $p(L)$, the *a priori* distribution over a set of locations $L=\{v_1, \dots, v_i\}$. The terminal nodes in the network represent “observable” variables that describe $p(E|L)$, the marginal conditional probability that you observe values $E=\{e_1, \dots, e_n\}$ from a sensor given that you are in location v in L . Given this network, we can get sensor readings $R=\{e_1, \dots, e_m\}$, for sensors 1 thru m , and calculate $p(L|R)$, the *a posteriori* probability distribution over L , the probability that we are in any location v_i given R . In general, any node in the network can be a query variable. For example, we could ask $p(s_1=e_1 | L=v_3, s_2=e_3)$, the probability that sensor s_1 will read e_1 given we are in location v_3 and sensor s_2 reads e_3 . The figure also includes a noise node that could affect the outcome of several sensors.

In the Bayesian network for determining location, values for L and E are quantized into discrete values. Nibble can theoretically distinguish n^s different locations in a Wi-Fi network where n is the number of output values from a sensor and s is the number of access points. However, the topology and construction of the building, as well as various noise effects, influence the actual number of locations Nibble can distinguish. For Boelter Hall, $n=4$ and $s=10$ and for FXPAL, $n=4$ and $s=4$.

Nibble has been used to calculate the probability that a laptop is in one of 14 adjacent locations 10-15 feet apart in an area covering two floors in Boelter Hall, a large building on the UCLA campus. In the current implementation of our system, $E=\{\text{High, Medium, Low, None}\}$, where $\text{None}=[0]$, $\text{Low}=(0,16)$, $\text{Medium}=(16,30)$, $\text{High}=(30,70)$ representing the ranges of signal-to-noise ratios (SNR) at each quantized level as measured between a laptop with a wireless card and an access point. These quantizations are based on early experiments at UCLA. Future implementations could have quantizations that are more sensitive to the exponential attenuation characteristics of the RSSI. It may be beneficial to have more levels with smaller ranges for quantizations at the higher end of the scale.

Training the Bayesian Network

To infer location using the Bayesian network in Figure 1, it is necessary to provide values for the *a priori* distribution $p(L)$ and the marginal distribution $p(E|L)$ for each access point. While it is possible to construct an analytical model to get the marginals, empirical measurements seem to work best [2]. For one of our sites, we took samples

of SNRs for 14 locations over several days. We sampled twice a day in the mid-morning and early evening. However, we observed that Nibble could correctly infer location with even just a few samples at each location.

Calibration can be tedious. Nibble does not require that all locations be declared *a priori*. Instead, L can be built incrementally. Locations can be added and deleted at anytime. Sampling occurs when a location is initially recorded and “re-sampling” can be done at anytime. Re-sampling allows the user to adjust the marginal distributions for an access point if there is a change that invalidates the current calibration, e.g. a change in the configuration of access points.

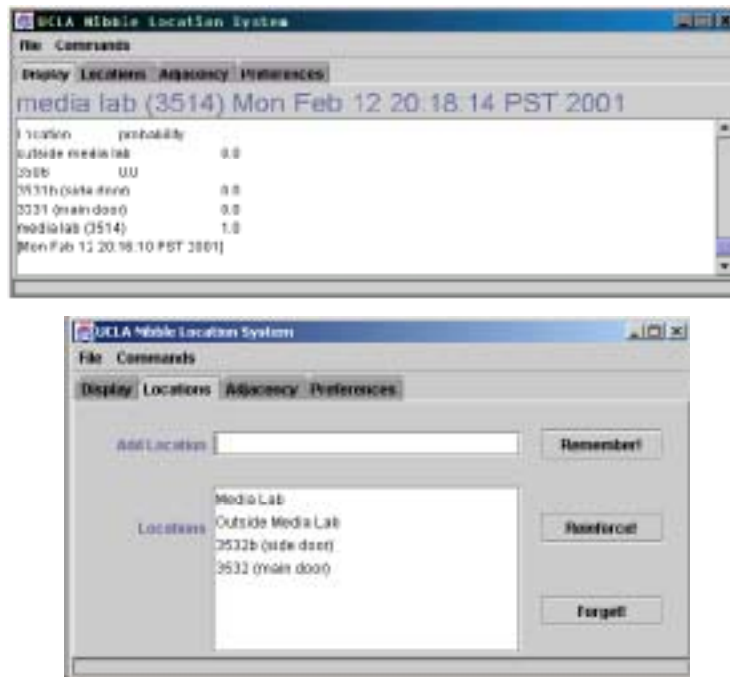


Fig. 2. Main Display of Nibble (top) and display for recording, calibrating, and deleting locations (bottom).

The default distribution for $p(L)$ is uniform. However, certain people are more likely to be in some locations rather than others. Profiles of people’s preferences for locations can easily be incorporated into the Bayesian framework by changing the *a priori* distribution $p(L)$. Although not part of the current implementation, Nibble could create a profile automatically by recording the frequency a client is at a certain location and updating the distribution over time. This same process could also be used to detect adjacency relationships between locations and used to more accurately pinpoint the location of a user. For example, it may be physically impossible for a transition to occur between locations v_1 to v_2 . Nibble models this similar to a Hidden Markov model where an additional node in the Bayesian network records the

transition probabilities for all locations. Currently, these adjacency relationships are recorded manually.

2.2 Using Nibble

Nibble is implemented using Java JDK 1.3 and currently works with Lucent Orinoco Wi-Fi cards running under MS Windows (see Appendix). The basic Nibble package includes a GUI that allows users to build and calibrate a location service as well as display and log the results of the fusion service. There is also a simple API that programmers can use to interface their applications to Nibble and make them location-aware (see Appendix).

Figure 2 shows the main display screen in the Nibble GUI while it is running. Nibble infers location from the SNR measures received from the Wi-Fi network card every two seconds. Nibble timestamps and then prints the resulting probability distribution over the set of locations in the large text area at the bottom half of the screen. The most likely location value is printed at the top of the text box. In the figure, Nibble is tracking 5 locations and the most likely location is the “media lab (3514).”

Adding a location can be accomplished at the “Locations” screen. It is useful to think of locations in two ways when using Nibble. First, there are distinguishable physical locations such as offices, closets, hallways, etc. Second, there are distinguishable signal locations where the signal quality measures are probabilistically consistent within a location. Nibble identifies signal locations rather than physical locations. These two types of locations do not necessarily map one-to-one but since physical characteristics of the space affect the signal measures there is a correlation. However, there may be physical locations that are indistinguishable from a signal perspective and conversely, there may be physical locations that encompass many signal locations.

To record a location, the user should go to that location and type in a name for it. When the user clicks the “remember” button Nibble will take signal quality samples at 1 per second for 10 seconds. When sampling is complete, the new location is added to the location list and is included in all future inferences. Nibble does not enforce any naming schemes so multiple labels can be given to the same location or multiple locations can have the same name. The latter is useful when there are many signal locations for one physical location. The former is useful if you want to add attributes to location labels such as “Media lab (evening)” and “Media lab (morning).” This assumes that the signal measures are different in the morning and evening.

If Nibble is unable to infer a location, it will display “???” as the most likely location. This usually occurs for locations that are “under-sampled” (or not calibrated at all). When a location is under-sampled, the initial 10 samples for a single location are not sufficient and the calculated marginal conditional probabilities are incorrect. Users can remedy this by “reinforcing” a location by selecting a location value in the

location list and clicking the “reinforce” button. Reinforcing is the same as sampling except new samples are added to modify the current marginals.

Reinforcement of locations may be insufficient to rectify incorrect location inferences particularly if there is a drastic change in the placement or configuration of access points. In this case, users can elect to delete locations from the location list using the “forget” button and build a new location model. Though not part of the current implementation, it is also possible to eliminate an access point from the model (e.g. if it moves). Locations can then be re-sampled and this will put the access point back in the model but with new marginals.

Nibble is mainly designed to be a component in a larger location-aware application. As such, Nibble has several interfaces that are available to application designers. For example, Figure 3 shows a simple graphing application that uses the Nibble API to get its location and display the probability distribution as a histogram. From the API, the application can start, stop, and read location data from Nibble. Nibble can also write a log file that applications can parse to get location data.

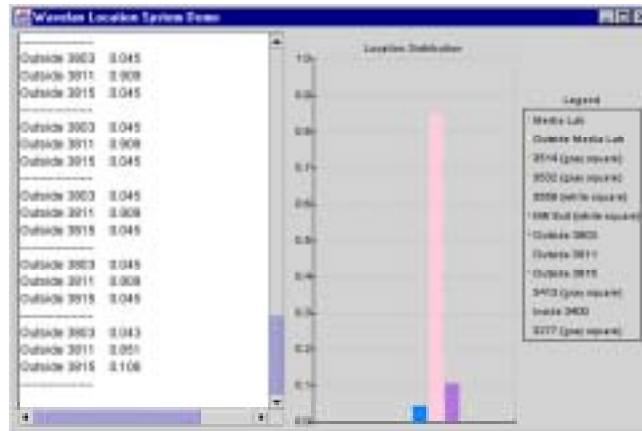


Fig. 3. Simple application that uses the Nibble API to display its current location as a histogram.

3. Experiences with Location Service

We have used Nibble at two research sites. The first site is a large engineering building on the campus of UCLA. The second site is an industrial research lab in Palo Alto, CA. Both sites differed in terms of their relative sizes, density of access points, and noise sources. For example, the campus building has a high density of access points to accommodate a very dynamic signal environment with many people

moving throughout the building at all times of the day. The industrial research lab had a relatively low density of access points with far fewer people in the building.

In this section we describe our experiences using Nibble in each of these facilities. Preliminary experiments were conducted on the campus to measure the accuracy of Nibble's location inferencing as well as its stability in the face of noise. Resource optimization was also considered. Exploratory usage tests were conducted at the industrial site and we report on the performance of Nibble in its support of a novel location-aware application.

3.1 Boelter Hall at UCLA

Boelter Hall is one of three engineering buildings on the UCLA campus. The hall is a large, 8-story building, which houses 4 engineering departments, the engineering library, and many classrooms. The building is square shaped, and in the center is an open-sky courtyard. During the daytime there are hundreds of students in Boelter Hall attending lectures and labs, and in-between classes the halls are filled with people. The Department of Computer Science is located on the third and fourth floors of Boelter Hall. Within the department there are 14 Wi-Fi access points that provide infrastructure support for many laptops.

The construction of the building causes severe attenuation of RF signal strength so a high density of access points is necessary. The current configuration allows connectivity to the wireless network in almost all locations in the department and even the courtyard. However, the density of access points varies in different parts of the building so some locations always have weak signals.

Hallway Tracking Tests

We conducted initial experiments to test the ability of Nibble to distinguish between different locations of a laptop as the user moves through the building at walking speeds. We conducted the tests primarily in the hallways of the building where connectivity was fair. To aid us in our experiments we implemented a distribution display that showed the probability distribution over locations as a histogram. We recorded 12 adjacent locations in the system and tested two different paths through these locations. There were a total of 10 access points that could be detected in these paths. The first path was a simple straight-line path from one end of the hallway to another. The second path went around the circumference of the building. This path went through different hallways and went from areas where the density of access points was high to areas where the density was lower. Part of the path also crossed an outdoor walkway.

We calibrated the system over a few days. For each location we took about 50 to 100 samples. In some locations we took fewer samples because we did not have access to the location after hours. Areas with weaker connectivity also prevented us from taking as many samples since our original implementation required connectivity to the

MUSE infrastructure [4]. This is not an issue with the current implementation of Nibble.

We measured the performance of the system by tabulating the correct, incorrect, and “don’t know” readings. Correct readings corresponded to when the most likely inferred location was the actual location as observed by the user. Incorrect readings corresponded to when the most likely location was not the actual location. “Don’t know” readings occurred if Nibble could not infer any location; this primarily happened in locations where we could not take enough samples. We defined an accuracy metric as the $\frac{\text{\#correct readings}}{\text{\#readings} - \text{\#don't knows}}$.

We experienced very good performance overall. Nibble achieved an accuracy of around 97% for the two different paths. The total number of “don’t know” readings accounted for about 15% of all the readings. However, these readings mostly occurred for the same parts in the building indicating either an under-sampled condition for those locations or a noisy space. For example, one of these spaces was near large metal fan ducts on an outdoor balcony.

The location service updates its inferred location every three seconds. As we changed our actual locations, the service sometimes took several inference cycles to update the most likely location to the correct one. This is most likely due to our sampling method where we concentrated mostly on the “center” of a physical space. We would generally have to come close to the center of the space for the location service to register the correct location. Later, we altered our sampling method to include samples from the boundaries (as perceived by the user) of the space. This generally produced more consistent results.

Several locations we recorded in the hallways of Boelter Hall could confuse the system. This is due to generally weak connectivity (all signal measures look similar) and noise. However, the system performed well in localizing our position in the hallway to several adjacent locations with one of those locations being correct. Even without adjacency information in the Bayesian network, the location service inferred we were in a location close to our actual location.

3.2 Research Lab at FXPAL

The FX Palo Alto Laboratory (FXPAL) resides on the second floor of a building that is rectangular in shape and measures 224 by 96 feet. There are about 40 offices, three small clusters of cubicles, and several conference rooms (see Figure 4). The FXPAL Wi-Fi environment consists of 4 WaveLAN access points and a number of wireless laptops. There are also 3 iPAQ Wi-Fi access points and wireless iPAQ PDAs; these have not yet been integrated with a location service.

Room Location Visualizer

We designed a room location visualizer and integrated it with the location data streamed out by Nibble. This visualizer is based on a blueprint of the building. The probabilities are quantized into 5 intervals: (0.0, 0.2], (0.2, 0.4], ..., (0.8, 1.0]. Rooms are shown in different shading levels according to their probability (see Figure 5). A subset of the rooms is targeted for each device, and the targeted rooms are shaded in light gray. Restricting to a subset of rooms increases the accuracy of the location service because this leaves fewer rooms in close proximity. Nibble updates the location probabilities every two seconds.

Accuracy

We tested the Nibble location service by having one of the FXPAL researchers carry a laptop whenever he went to a meeting or to work in another room other than his own office. There are 9 room locations that are frequented by Patrick: 3 conference rooms, 2 collaborators' offices (cubicles), his own office, his manager's office, a lounge, and a patio outside the building near a coffee area.

We found Nibble always able to distinguish between the 3 conference rooms. The probabilities usually fell in (0.6, 0.8] and (0.8, 1.0]. One factor is that the conference rooms are not adjacent to each other. The nearest two are diagonally across a hallway. Another factor is that conference rooms are larger than offices and cubicles.

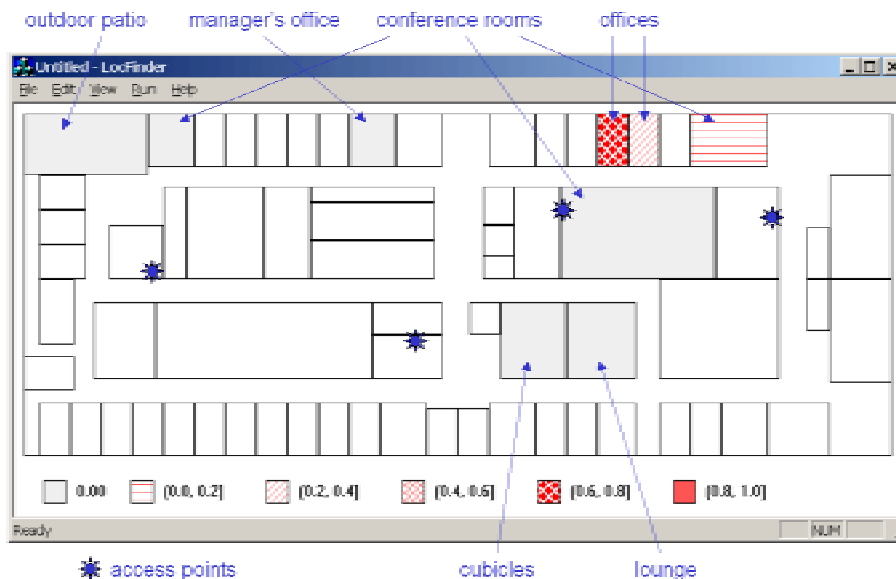


Fig. 4. Room location visualizer with floor plan of FXPAL.

Sometimes Nibble was confused between adjacent offices (dimensions are 8 x 14 feet). The probabilities usually fell in the range (0.4, 0.6]. Nibble is able to distinguish offices that are separated by another office, with probabilities in (0.8, 1.0]. In our evaluation, this level of accuracy was sufficient, because the researcher did work on his laptop in his office and not his neighbor's.

Cubicle clusters tested the limits of Nibble. Nibble was able to locate a cluster, but not able to distinguish between the cubicles within a cluster. A cubicle's dimension is about 6 x 8 feet, with 2 to 4 cubicles per cluster. The clusters are dispersed far from each other on the building floor. Like the uncertainty caused by adjacent offices, accuracy can be improved for cubicles in the same cluster if we know which cubicle in that cluster a user and his laptop are more likely to frequent.

A Room-Aware Application at FXPAL

An application at FXPAL that makes use of room location is a multimedia meeting minutes system called *LiteMinutes* [5]. The main conference room at FXPAL is equipped for multimedia meeting capture with digital video, presentation slide images, and notes [5]. See Figure 5. Occasionally, video of meetings is recorded in other conference rooms, and we are considering plans to capture teleconferences between FXPAL and Fuji Xerox in Japan.



4. Toward an Infrastructure for Wi-Fi Room-Aware Applications

Nibble is derived from a location service implemented within MUSE, an infrastructure designed to support a wide-range of sensor-based services. In this section we discuss how the fusion service model can be utilized to construct component-based location services that potentially offer better performance and more flexibility for application designers.

4.1 Categorizing Location Systems

Researchers have been interested in indoor locations systems for at least the past decade. Systems generally fall into two categories: 1) *tracking systems* that provide location information from a central infrastructure and 2) distributed *beacon systems* that allow a client to infer location from beacons they see in the environment.

In the first case, the location system depends on identification messages from tracked objects that get processed in an infrastructure specific way to infer the location of the client. For example, the Xerox ParcTab and Active Badge system can localize a user to a room based on the physical location of the receiver node that detects client's identification message [18][17]. More precise location systems such as BATS and 3D-iD use arrays of RF sensors capable of locating specially tagged objects in three-dimensional space within centimeters [8][13]. User identification is also possible in tracking systems. The SmartFloor project can identify and track a user using Hidden Markov models based on "footfall" signatures acquired from a floor equipped with load sensors [11].

In the second case, the environment provides beacons from which a client can infer its current location. Early examples of this include the CyberGuide that used IR beacons as proximity sensors [1]. A more recent example is the Cricket location system from M.I.T. [14] that uses RF and ultrasonic beacons that send out location identification strings that can be detected by a mobile user.

At first glance, a Wi-Fi –based location system such as Nibble is a beacon system. This is not necessarily true. For example, it is possible to implement a process at each access point that collects all SNR data from clients. Then a backend process could infer the location of any Wi-Fi client by considering all access point data. RADAR was implemented in this way [2]. One consequence of this implementation is that devices can easily know the location of other devices in the network. Privacy in this case is an important concern. The current "beacon" implementation of Nibble allows the device to be "invisible" as long as it just monitors the network and does not transmit any network packets.

4.2 Component-based location systems

The fusion service model used in Nibble is not limited to a Wi-Fi indoor location system. Other location systems are compatible with probabilistic formulations. For example, Cricket beacons use radio frequency (RF) and ultrasonic signals to broadcast a string to nearby clients that identifies the current spatial location of the client. Since multiple beacons can be heard, the client assumes that the beacon that it is closest to identifies the correct location. One useful assumption is that the beacons are only placed near entranceways or partitions between spaces; this ensures that the closest (and hence correct) beacon can be identified with a high probability.

A fusion service utilizing Cricket beacons does not require exact placement of beacons though the trade-off is the need to calibrate the Bayesian network to get the marginals. For Cricket, the beacon values can be quantized as $E = \{\text{Heard and Closest, Heard, Not Heard}\}$, and the fusion service would calculate the $P(L|R)$, the probability that it is in some location in L given that it observes $R = \{E_1, \dots, E_m\}$ from the Cricket beacons. This potentially increases the resolution of Cricket from s locations to 3^s locations since the original system requires one beacon per location. Given the limited range of the beacons, however, at anytime you will see less than s beacons at a location.

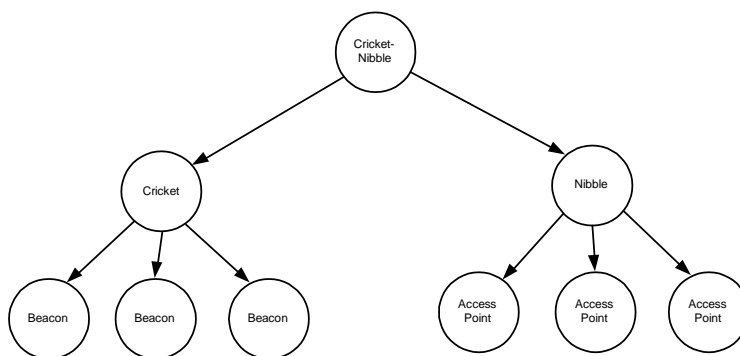


Fig. 7. Composing two location services

One distinct advantage of the fusion service approach is the ability to compose services together. Composability is a major reason MUSE adopts a general model of data aggregation and interpretation. For example, Figure 7 shows a possible Bayesian network that combines Cricket and Nibble into a new Cricket-Nibble fusion service that utilizes both. The Cricket-Nibble service can be used by applications in place of just Nibble or Cricket since the output is still a probability distribution over locations, and the resource optimizations possible in Nibble can be applied to Cricket-Nibble.

Composability is a step towards providing different degrees of privacy for a user of a location system. For example, a user may allow less privileged users access to coarse location information (e.g. the building level) about her and allow a restricted set of users higher resolution data (e.g. the room level). To set up this partition the infrastructure allows different permissions on accessing each fusion service where some users can only get “low quality” information while other users get “high quality” information.

Characterizing Information Quality

In many physical sensor systems, power or communication bandwidth is highly constrained [10]. Having a quantifiable measure of quality for a fusion service allows applications the ability to choose between fusion services based on a unified measure of expected performance for a given “cost.” For example, there may be two different fusion services capable of inferring location but one service may consume a lot of power or bandwidth and be 90% accurate while another is only 75% accurate but consumes far less resources.

We define the term “Quality-of-Information” (QoI) to characterize the performance of a fusion service in terms of: 1) accuracy of the inference as measured by an outside observer and 2) the level of confidence it has in its measure [4]. For example, a uniform distribution over all possible locations is an undesirable result. The best result would be an “impulse” for a single value v in L and $p(v)=1.0$. We capture this intuition using the information theoretic concept of entropy, which we approximate by the probability of the most likely outcome of the query variable. Intuitively, a fusion service that infers a most likely location with $p=90\%$ is better than a service that infers a most likely location with $p=50\%$. Note that there is a difference between the “confidence” of the system and the “resolution” of the system. A location service that can only distinguish between 2 large locations (e.g. 2 floors in a building) is not necessarily worse than a location service that can distinguish many small locations. If both are always 90% sure of their inferences and are 90% accurate, then we say they have the same QoI regardless of their resolution differences. Of course, someone may prefer higher resolution location information but QoI is a measure relative to the desired or specified resolution of the “query variable” (e.g. location).

Given a measure of quality, we can pose an optimization problem: achieve a high QoI at minimal cost. This is similar to the “trouble-shooting” optimization problem in [9]. In Nibble, we assume a unit cost for consulting an access point and now we would like to get accurate location information by consulting the minimal number of access points. Our approach is to consult one access point at a time until we reach a user specified level of QoI or we run out of access points. We consult access points that contain the “most information” about location before we consult access points with “less information.” To measure the amount of information we can potentially get from consulting an access point we use a heuristic based on an off-line sensitivity analysis of the Bayesian network [4]. Note that this resource optimization approach can be generalized to all fusion services.

Optimizing QoI for a Wi-Fi Location Service

To test our ideas we used an enhanced version of the location service where the client can specify a desired level of quality for the inferred location information. The fusion service would attempt to achieve this quality but consult the minimal number of access points. For example, the client could specify a quality setting $Y=0.9$ which means that the most likely location should have a probability of 0.9. The location service will consult access points (ordered by our heuristic) one at a time until it achieves this quality setting.

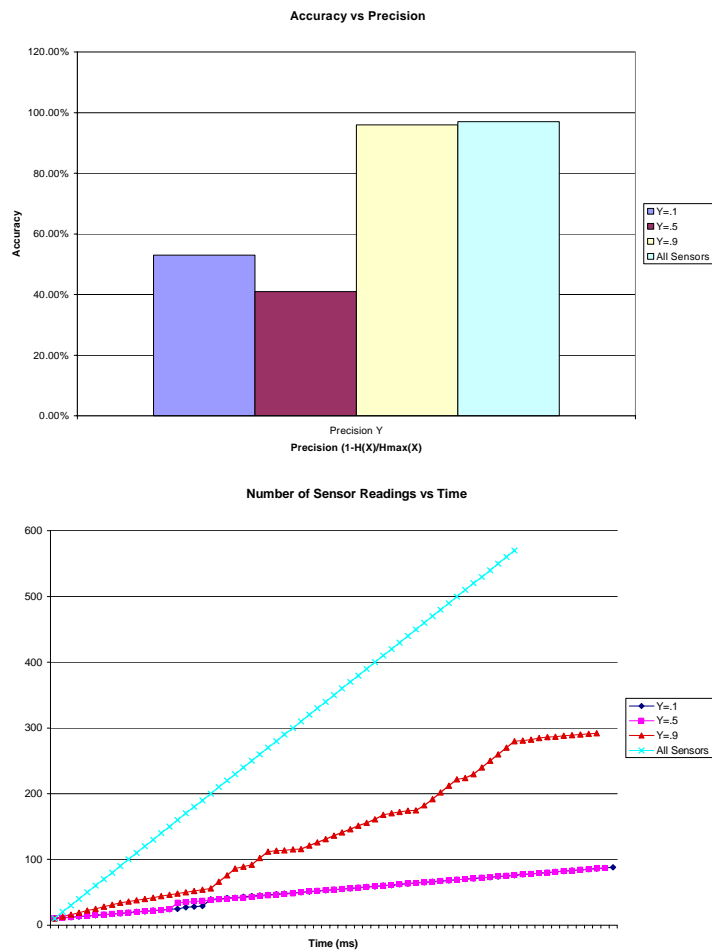


Fig. 8. Accuracy vs. QoI (left) and Cost vs. QoI

Figure 8 shows the effect on accuracy and cost for quality settings of $Y=\{0.1,0.5,0.9,ALL\}$ where ALL means consult all access points. For a quality setting of $Y=ALL$, the location service was 97% accurate but it was the most expensive since it consulted all access points for each inference. For a quality setting of $Y=0.9$, the location service was 96% accurate but it was considerable less expensive because it only consulted around three access points for each inference. For the other settings, the location service was about 50% accurate.

5. Conclusions and Future Work

In this paper we described our experiences using Nibble, a probabilistic location service for a Wi-Fi networked environment. We explained the MUSE sensor fusion model and how it is possible to build component-based fusion services to infer location. We are currently implementing an enhanced version of Nibble that infers location using a heterogeneous set of sensors. The UCLA Multimedia Systems Laboratory has constructed a small, inexpensive sensor board that can communicate with a PC through a standard serial port. The sensor board is capable of measuring temperature, humidity, orientation, and acceleration. The inclusion of this sensor data can improve the overall performance and resolution of Nibble. As an example, oftentimes when a user enters a room they will place their laptop in a specific location. If two locations are nearly indistinguishable from Wi-Fi measurements but the laptop is placed at each location with different orientations, we can use the digital compass to get the laptop location. This is also useful if we need to discriminate locations within a single physical space. Similarly, temperature and humidity can be used to distinguish colder and warmer environments (e.g. a machine room versus an office). The more general problem of “grafting” together services in an automated fashion is a topic of research we plan to investigate.

References

1. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., and Pinkerton, M. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3 (1997), pp. 421-4333.
2. Bahl, P., Padmanabhan, V. RADAR: An In-Building RF-based User Location and Tracking System. In *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000. IEEE Computer Society Press.
3. Brown, P.J. The stick-e document: a framework for creating context-aware applications. In *Proceedings of Electronic Publishing '96*, pp. 259-272.
4. Castro, P. and Muntz, R. Managing context for smart spaces, *IEEE Personal Communications*, vol.7, no. 5, October 2000.
5. Chiu, P., Boreczky, J., Girgensohn, A., Kimber, D. LiteMinutes: An Internet-based system for multimedia minutes. *Proceedings of Tenth World Wide Web Conference (2001)*.
6. Chiu, P., Kapuskar, A., Reitmeier, S., and Wilcox, L. Room with a Rear View: Meeting Capture in a Multimedia Conference Room. *IEEE Multimedia Magazine*, vol. 7, no. 4, Oct-Dec 2000, pp. 48-54.

7. Dey, A. *Providing Architectural Support for Building Context-Aware Applications*. PhD Thesis. Georgia Institute of Technology, Department of Computer Science.
8. Harter, A., Hopper, A., Steggles, P., Ward, A., Webster, P. The Anatomy of a Context-Aware Application. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, Washington, USA, August 1999. ACM Press.
9. Heckerman, D., Breese, J., Rommelse, K. "Troubleshooting Under Uncertainty," *DX-94-5th Int'l. Workshop on Principles of Diagnosis*, AAAI, 1994, pp. 121-130.
10. Intanagonwiwat, C., Govindan, R., Estrin, D., Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks*, Boston, MA, August 2000. ACM Press.
11. Orr, R., Abowd, G. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In *Proceedings of the 2000 Conference on Human Factors in Computing Systems*, The Hague, Netherlands, April 1-6, 2000.
12. Pearl, J. *CAUSALITY: Models, Reasoning, and Inference*, Cambridge University Press, 2000.
13. PinPoint, <http://www.pinpointco.com>
14. Priyantha, N., Chakraborty, A., Balakrishnan, H. The Cricket Location Support System. In *Proceedings of the Sixth Annual ACM International Conference on Mobile Computing and Networking*, Boston, MA, August 2000. ACM Press.
15. Schilit, B., Adams, N., and Want, R. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994. IEEE Computer Society.
16. Small, J., Smailagic, A., Siewiorek, D. Determining User Location For Context Aware Computing Through the Use of a Wireless LAN Infrastructure. Project Aura report. <http://www.cs.cmu.edu/~aura/docdir/small100.pdf>
17. Want, R., Hopper, A., Falcao, V., Gibbons, J. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91-102, January 1992.
18. Want, R., Schilit, B., Adams, N., Gold, R., Petersen, K., Goldberg, D., Ellis, J., Weiser, M. The ParcTab Ubiquitous Computing Experiment. Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995. Also appears In *Mobile Computing*, H. F. Korth and T. Imielinski, eds., Kluwer Academic Press, 1996.

Appendix: Getting Nibble

Nibble is available for download and experimentation at <http://mmsl.cs.ucla.edu/Nibble>. Applications can interface to Nibble through a simple API. The API is not designed to support the construction of models; this should be done through the GUI. Instead, the API allows users to activate and de-activate Nibble as well as access the results of its location inferencing. Figure A shows a Java code snippet that applications can use to access location data from Nibble.

```

Nibble nibble = new Nibble();
nibble.on();

/** wait a bit since Nibble calculates in 3 second
cycles..... */

```

```
mywaithere(5000); // application defined (wait 5
seconds)

/** .. then get the data */

String location = nibble.getLocationName();
String distribution = nibble.getDistribution();

nibble.off();
```

Fig. A. Java programming example to access location data in Nibble.