Virtual Compass: relative positioning to sense mobile social interactions

Nilanjan Banerjee
University of Arkansas, Fayetteville

Sharad Agarwal
Microsoft Research

Paramvir Bahl
Microsoft Research

Ranveer Chandra
Microsoft Research

Alec Wolman
Microsoft Research

Mark Corner
University of Massachusetts, Amherst

January 2010

Technical Report
MSR-TR-2010-5

There are endless possibilities for the next generation of mobile social applications that automatically determine your social context. A key element of such applications is ubiquitous and precise sensing of the people you interact with. Existing techniques that rely on deployed infrastructure to determine proximity are limited in availability and accuracy. Techniques that rely on specific frequencies or signal processing are not supported by widely deployed mobile devices. Virtual Compass is a peer-based relative positioning system that relies solely on the hardware and operating system support available on commodity mobile handhelds. It uses Wi-Fi and Bluetooth radios to detect nearby mobile devices and places them in a two-dimensional plane. Virtual Compass simultaneously uses multiple radios and multi-hop relaying to reduce distance estimation error and increase coverage. We use adaptive scanning and out-of-band coordination to explore trade-offs between energy consumption and the latency in detecting movement. We have implemented Virtual Compass on mobile phones and laptops, and we evaluate it using a sample application that senses social interactions between Facebook friends. Our experimental evaluation shows high accuracy and low latency with modest energy consumption.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

1 Introduction

Imagine a suite of social applications running on your mobile phone which senses your precise social context, predicts future context, and logs and recalls social interactions. The possibilities for such applications are myriad [33], from alerting you about an impending contact with a business associate and reminding you of their personal details, to a game that utilizes the relative physical positioning of its players, or a service that tracks the frequency and tenor of interactions among colleagues and friends. These next generation applications will use continual sensing of social context at an extremely fine granularity. Recent examples of mobile social applications include Loopt [4] which displays the location of a user's friends and Dodgeball [2] which finds friends of friends within a 10 block radius. Unfortunately, these and other widely deployed technologies that implement localization on mobile handhelds are limited by accuracy, coverage and energy consumption.

The most widely used localization technology in mobile handsets is GPS, but it rarely works indoors. Furthermore, its accuracy degrades in urban environments, and the energy consumed by GPS devices is a significant deterrent. Cell-tower based localization [38] is widely available but can provide very poor accuracy without a fingerprint profile, or outside city centers. Wi-Fi localization, when available, provides reasonable accuracy in dense urban environments, but is also much less effective in other areas [30].

People spend the majority of their time indoors. As a result, many of the most common opportunities for social interaction occur in indoor environments such as offices, hotels, malls, restaurants, music and sports venues, and conferences. In these environments, to detect the interaction with, or even the opportunity to interact with someone requires relatively fine-grained location accuracy. Even in environments where indoor Wi-Fi based localization schemes [21] could provide the needed coverage and accuracy, most of today's environments do not have this infrastructure deployed and the barriers to deployment lead us to believe that this will be the case for some time to come. Techniques that rely on ultrasound or detecting the phase offset of transmitted radio waves [32] are difficult to implement using the hardware and APIs available on commodity mobile phones.

We present the design and implementation of Virtual Compass, a peer-based localization system for mobile phones. Virtual Compass does not require any infrastructure support, but instead uses multiple, common radio technologies to create a *neighbor graph*: a fine grained map of the relative spatial relationships between nearby peers. Virtual Compass allows nearby devices to communicate directly, and provides multi-hop relaying so that the neighbor graph can include others who are not within direct communication range.

Virtual Compass leverages short-range radio technologies, such as Bluetooth and Wi-Fi, available in today's mobile handhelds. These radios consume a significant amount of energy during scanning, and we consider energy management as a fundamental design pillar. Hence, Virtual Compass includes three techniques to reduce energy consumption: 1) use of adaptive scanning triggered on topology changes to update the neighbor graph; 2) selection of the appropriate radio based on its energy consumption characteristics; and 3) using the wide-area wireless network when available with a cloud-based service to assist with coordination and notification of potential changes to the neighbor graph.

Mobile social applications are heavily driven by the *relative* positioning of people, and less by *absolute* location. Sensing the precise placement of individuals relative to one another yields the social context needed for many useful applications, and the quality of location information produced by Virtual Compass increases as the density of devices increases. Nonetheless, Virtual Compass can opportunistically leverage infrastructure resources, such as cell-tower information and GPS, when they are available, to convert the relative map to an absolute one.

We have implemented Virtual Compass on Windows based mobile phones and laptops. Through extensive experimentation we evaluate the latency, location accuracy, and energy consumption characteristics of Virtual

Compass as a function of system scale. In a typical experiment we found the average error in spatial placement of nine nodes in a $100m^2$ area was 1.9 meters and the worst case error was 2.0 meters. We show significant accuracy gains in simultaneously using multiple radios for distance estimation, and our algorithm for spatial placement. Additionally, we are able to locate a new device within 25 secs of its arrival. Applying our energy conservation techniques yields four-fold to seven-fold improvements in battery lifetime, depending on the scenario. We also present the design and evaluation of a sample application built on top of Virtual Compass.

2 Related Work

As a key ingredient for sensing, localization has been the subject of extensive work, both core technologies, and systems that leverage and reason about location. A comprehensive review of localization research is in [31]. Here, we compare and contrast our work by broadly dividing the corpus of prior work into two categories: infrastructure-based and peer-based, and review the most relevant in each.

Infrastructure-based localization techniques can be broadly classified by their core technology: GSM [27, 29, 30], Wi-Fi [3, 13, 21, 28], GPS, ultrasound with RF [14, 36, 41], Infrared [25, 40], RFID [11], and UWB [10]. While each technology demonstrates a cost benefit trade-off, we contend that systems will use widely available hardware found in commercial mobile devices. In fact, the most successful techniques have leveraged infrastructure that was put in place for other reasons (GSM and Wi-Fi localization) and it seems likely that peer-based localization will follow a similar trend relying on technologies such as Wi-Fi and Bluetooth. GPS is the only exception, but it is unique in that it only works outdoors—the reason for its cost effectiveness. Several industrial startups [2, 4, 7, 9] have cropped up which use localization to support social applications, relying on the infrastructure-based localization support in mobile phones which is typically Wi-Fi-, GSM- or GPS-based.

However, such schemes are limited in coverage and accuracy, making it impossible to support the full range of social applications—especially in situations that require fine-grained proximity information. For example, Wi-Fi localization requires a dense deployment of access points and accurate profiling (not available in many indoor scenarios), and GSM localization can exhibit poor accuracy without a detailed profile or away from dense urban areas. Our experiments using the popular iPhone Map application across a university produced a mean accuracy of around a *kilometer*—typically four GSM cell towers were visible. GPS-based localization on the other hand does not work indoors and custom infrastructure deployment is often infeasible.

Peer-based localization techniques attempt to either infer the proximity of a pair of devices, or infer the actual distances between multiple devices and place them in a virtual map.

Proximity-based placement schemes such as Hummingbird [24] and NearMe [26] detect if two devices are within 30 to 100 meters of each other. Beep Beep [35] achieves high accuracy using sound, but does not spatially place more than two nodes, nor nodes that are out of earshot. BlueHoo [1] uses Bluetooth discovery to detect friends within Bluetooth range.

Instead, Virtual Compass measures the distances between multiple nearby nodes, uses multi-hop communication to expand coverage, and spatially places them relative to each other on a 2D plane. Moreover, our system uses algorithms which balance energy consumption with low-latency, accurate localization, which has also been explored for infrastructure-based techniques [19].

Relate [22] and DOLPHIN [23] rely on custom ultrasound hardware which is typically unavailable in commodity devices. RIPS [32] requires signal processing of received radio waves, which is possible on custom hardware such as MICA2 motes but hard to do with off the shelf mobile phones and standard SDKs.

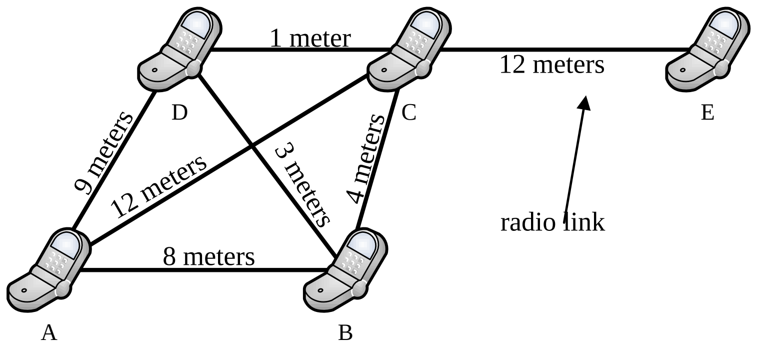

Figure 1: A set of mobile nodes. Each line represents the ability to directly communicate between the two end-points using a radio such as Wi-Fi or Bluetooth. A,B,C and D are in communication range of each other, while E is only in range of C.

MSP [44] uses sensor event distribution to locate nodes in a static sensor network. Bulusu [15], Sextant [20] and Calibree [39] use the location of a subset of nodes (e.g. equipped with GPS units) to derive the locations of a larger set of nodes. LOCALE [43] also uses GPS equipped nodes to help other nodes using dead reckoning. Our goal is to design a peer-based localization system that works in mobile scenarios in the absence of fixed infrastructure or reference points, which can be hard to obtain using GPS in indoor settings.

Moore et al. [34], Spotlight [37], and Vivaldi [18] address the problem of placing nodes relative to each other in a multi-dimensional plane. Moore et al. [34] use the idea of robust quadrilaterals and three different phases of cluster localization, cluster optimization, and transformation to place ultra-sound equipped sensors relative to each other. While their scheme can be used in Virtual Compass, we adopt a simpler scheme (a variant of Vivaldi [18]) and we concentrate on commodity cell phone hardware. We use multi-radio composition and multi-hop relaying in for better accuracy and coverage. Moreover, our study involves several techniques for energy management on commodity cellular phones. Spotlight [34] uses space time properties of events in the network to place nodes relative to each other. While Spotlight uses MICAz and XSM motes for evaluation, Virtual Compass focuses on commodity hardware on cellular phones and widely available radios such as Bluetooth and Wi-Fi.

3 Peer Localization

The goal of Virtual Compass is to generate a two-dimensional layout of nearby mobile devices. It uses radios that allow peer-to-peer communication, such as Wi-Fi and Bluetooth, to exchange messages directly between devices. This exchange serves two purposes. Each pair of devices that are in communication range uses the received signal strength of these messages to estimate the distance between them. The message itself contains the list of neighbors and their distances, which allows nodes that are further away to map devices that are not in their immediate communication range. Virtual Compass leverages the collective knowledge of distances between peers learned in this way to calculate the 2D layout.

Figure 1 shows an example. Mobile node A periodically sends messages to its neighbors B, C, and D. Each of these nodes uses the received signal strength indication (RSSI) of these messages to calculate its distance to A, as described in § 3.1. We exchange these messages on multiple radios to reduce the inherent

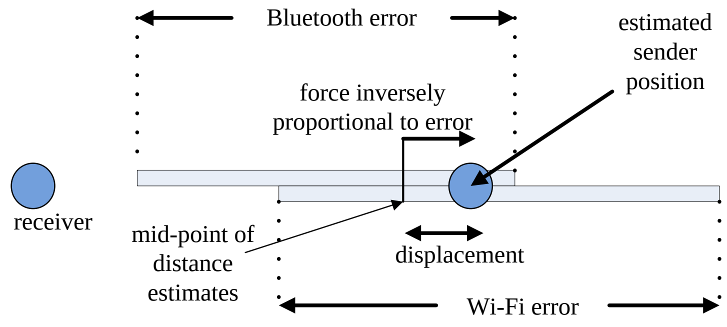
Bluetooth error
estimated sender position
force inversely proportional to error
receiver
mid-point of distance estimates
displacement
Wi-Fi error

Figure 2: An example of using RSSI measurements from multiple radios (Bluetooth and Wi-Fi) to reduce the error in computing distance.

error of distance estimation via RSSI, as described in § 3.2. We embed these distances in the messages that are exchanged between neighbors so that each node discovers the distances between other nodes. So in this way, C learns of the distance between A and D. Furthermore, nodes such as E that are far away can learn where A, B and D are. Virtual Compass solves the constraints imposed by these distances to create a relative map using the technique in § 3.3. Note that the underlying RSSI-based mechanism detects distance but not direction.

3.1 Estimating Distance

In Virtual Compass, nodes periodically exchange messages on radios with omni-directional antennas. § 5 describes these messages in detail. We use the RSSI of these messages to estimate the distance between sender and receiver. Even though we rely on RSSI, if techniques such as *propagation time* become feasible, Virtual Compass can easily use them instead. While translating RSSI to distance has been studied in prior work [42, 17], Virtual Compass enhances that work by incorporating the uncertainty in distance measurements to provide two benefits. First, as Virtual Compass is meant to be used in a broad range of unknown environments, modeling the uncertainty reduces the dependence on the environment in which the measurements were taken. Secondly, and more importantly, the error model provides a basis for composing information from different radios, as we show next.

To translate each RSSI reading to a distance estimate with an error bound, we use empirical models that we built by running several propagation experiments. Similar to prior work, we placed devices at known distances and measured RSSI for multiple packets over both Bluetooth and Wi-Fi. We used different types of mobile phones and laptops. For each RSSI value in each radio technology, we calculate the mean distance, and the uncertainty. We experimented with several ways to calculate uncertainty, including inter-quartile range and max-min, and found that the difference between 90^{th} and 10^{th} percentile distances performs best in practice. We use regression to model the distance and error for these ranges. We present the empirical models in § 5.

3.2 Using Multiple Radios

To reduce the error in estimating distance from RSSI, Virtual Compass uses multiple peer-to-peer radios simultaneously. For ease of exposition, we describe how our scheme works for two radios, Wi-Fi and Bluetooth.

This approach works for any radio with an RSSI to distance conversion, or when using more than two radios.

Consider Figure 2 where a node receives a message from the sender over Bluetooth and one over Wi-Fi and attempts to calculate the distance between the two nodes. Let $RSSI_1$ be the RSSI of the message received over Bluetooth, and $RSSI_2$ be the RSSI of the message received over Wi-Fi. As described in § 3.1, we obtain a distance estimate for each, x_1 and x_2. We also obtain the uncertainties, u_1 and u_2, each of which is the distance between the 10^{th} and 90^{th} percentiles. The goal of the composition is to combine the two sources of information in order to reduce uncertainty in measurement. The mid-point of the two RSSI distance estimates is $P = (x_1 + x_2)/2$. We apply a displacement from P for each measurement, which are $F_1 = (P - x_1) * u_1/2$ and $F_2 = (P - x_2) * u_2/2$. Intuitively, the sum of the forces should push the node in the direction of a source which has a smaller uncertainty in measurement. The final estimate of the distance is given by the midpoint displaced by a normalized sum of displacements $D = P + 2(F_1 + F_2)/(u_1 + u_2)$. The normalization ensures that the estimate of distance always falls within the range of estimates given by the two RSSI readings. In the rare case where the uncertainties from the two readings do not intersect, we simply use P as the final distance.

In this way, each pair of nodes that can directly communicate with each other estimate the distances between them, while reducing error. These distances are embedded in the messages that are exchanged between them so that ultimately, each node knows the distances between any two nodes that can communicate in the vicinity. The next step in Virtual Compass is to calculate a 2D spatial placement of these nodes that satisfies these distance constraints.

3.3 Spatial Placement

Consider a 2D Euclidean space where each node's position is determined by its (x, y) coordinates. Each distance estimate r_{ij} between nodes i and j forms a constraint: $(x_i - x_j)^2 + (y_i - y_j)^2 = r_{ij}^2$. An optimal algorithm would simultaneously solve this set of *non-linear* (quadratic) constraints to calculate coordinates for peer nodes. However, this is known to be NP-Hard. Furthermore, since the distances between nodes are measured independently and are subject to error, it is possible in some cases that there is no solution that satisfies all of the distance constraints.

We instead use the Vivaldi [18] method to calculate node positions. Vivaldi uses estimates of distances between nodes to calculate a force vector, and then iteratively improves each node's coordinates by moving it along the resulting force. Vivaldi has been shown to produce good results with little computation overhead. However, the choice of starting all nodes at the origin can sometimes lead to local minima or a large number of iterations to converge. Hence, to produce a relative map of all nodes, we first calculate a very approximate but quick placement in *phase 1*, and then feed that to a simple Vivaldi implementation in *phase 2* for iterative refinement.

Phase 1 calculates an approximate set of coordinates that will help Vivaldi converge faster and to more accurate results in phase 2. The algorithm we use is defined in Algorithm 1. Consider the example in Figure 3, where node A is calculating a placement for itself with respect to 3 other nodes, and begins by placing itself at the origin. It finds the peer, B, that is the smallest distance (r_1) away, and places it at (0, r_1). Since our underlying distance estimation is unable to discern direction, this placement is arbitrary as long as it is r_1 distance from the origin. Next, we choose node C because it is constrained by both A and B. There are multiple solutions for placing C since the constraints are quadratic, and we randomly choose one. Next, we again choose another node D that is constrained by as many of the currently placed nodes as possible. Again, there are multiple solutions and we select the set of locations that is the closest together and take an average between these coordinates to place D. We run this algorithm multiple times with different constraint

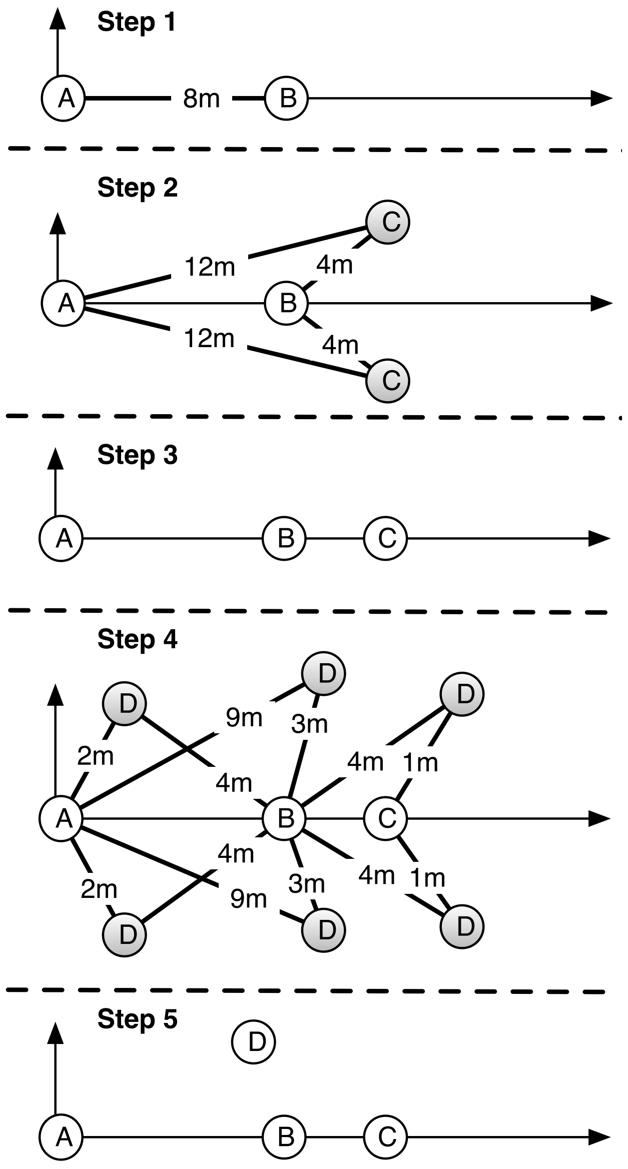

Figure 3: An example of spatial placement in Phase 1

orderings and we use an average of the coordinates from each iteration as the starting placement for phase 2. Experimentally, we determined that 10 iterations produces a sufficiently accurate initial placement with little impact on run time. While we could have used other algorithms, the goal of this phase is to produce a starting point for Vivaldi that is more reasonable than the origin for all nodes.

Algorithm 1 Spatial placement for calculating rough 2D coordinates during Phase 1 in Virtual Compass

```
Input: Set of constraints $C = \{C_1, ..., C_n\}$
Constraint $C_i$ of form $(x - x_i)^2 + (y - y_i)^2 = r_i^2$
loop
    For every pair of constraints $(C_i, C_j)$
    Find intersection points $(x_1, y_1)$ and $(x_2, y_2)$
end loop
$P$: set of coordinate pairs from constraint intersection.
$P = \{\{(x_1^1, y_1^1), (x_2^1, y_2^1)\}, ..., \{(x_1^k, y_1^k), (x_2^k, y_2^k)\}\}$.
{Initialize solution set with coordinate of first pair chosen randomly.}
Intersection set $S = \{(x_1^1, y_1^1)\}$
loop
    For each element $E = \{(x_1^j, y_1^j), (x_2^j, y_2^j)\} \in P$
    {Find intersection point which has minimum distance with points in S}
    $S = S \cup \arg\min \sum_{(x_j, y_j) \in S} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
end loop
return Node coordinate: $(\frac{1}{|S|} \sum_{x_i \in S} x_i, \frac{1}{|S|} \sum_{y_i \in S} y_i)$
```

Phase 2 uses the coordinates from phase 1 as the starting placement and uses a simple implementation of Vivaldi [18] to iteratively refine the coordinates to reduce the error between the placement and the measured pairwise distances. In each iteration, Vivaldi calculates forces that are applied between nodes – each force represents the difference between the measured distance between a pair of nodes and their distance in the virtual coordinate space. The resulting force on each node then determines the direction and amount of movement for the node in the virtual coordinate space. This process is repeated in each iteration. We have experimentally determined that 100 iterations produces accurate results with extremely marginal benefit from additional iterations. In § 6, we present the latency overhead of this computation, and it is dwarfed by the network communication time.

As an example, consider node A at (x_1, y_1) with a neighbor B whose coordinates are (x_2, y_2). The measured distance between them is r_{12}. The magnitude of the force F between them as applied on A is $r_{12} - \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ and its direction is given by the unit vector $((x_1 - x_2), (y_1 - y_2))$. There may be other forces applied on A (due to measured distances to other neighbors), and we calculate the resulting single force [18]. The coordinates for A are then changed in this iteration to $(x_1 + F_x * t, y_1 + F_y * t)$, where F_x and F_y are the components of F in the x and y direction and t is a constant. For Virtual Compass we experimented with different values of t and found $t = 0.1$ works best in our environment. Applying a force at each node that is proportional to the error in distance between node pairs minimizes the mean-square error and converges to a set of virtual coordinates which best satisfy the distance constraints (see [18] for proof).

3.4 Geographic Location

Virtual Compass spatially places peers relative to each other in a 2D plane. Our underlying use of pairwise RSSI distance estimation is well suited to this task. However, if GPS or any other infrastructure-based

Radio	Power (mW)	Lifetime (hours)
GSM (idle)	24.4	203.0
Bluetooth (idle)	45.5	109.8
Bluetooth (scanning)	507.6	9.8
Wi-Fi (idle)	849.9	5.9
Wi-Fi (scanning)	1305.4	3.8
GPS (idle)	859.9	5.8
GPRS (transfers)	1031.2	4.8
HSDPA (transfers)	1099.6	4.5

Table 1: Energy consumption of radios on fully-charged HTC Touch Cruise phones. Each row is the result of a separate experiment with only one radio on.

localization is opportunistically available, Virtual Compass uses it to translate the virtual coordinates to latitude and longitude coordinates. We need at least three nodes to report their absolute location. Virtual Compass pins the spatial location of these nodes using their geographic coordinates in place of their virtual coordinates, and then correspondingly the remaining peers' coordinates are translated.

4 Energy-Efficient Peer Localization

As with any system targeted at mobile devices, energy consumption is a critical concern. If the lifetime of the device is severely impacted, users will eschew applications that rely on our system. Virtual Compass depends on frequent communication between peers to provide timely updates to changes in the social graph. As has been observed in prior work [19], communication consumes a significant portion of a mobile phone's energy budget. To place our work in a common frame of reference, we include Table 1 which shows the energy consumption of our implementation platform. With no communication, a typical phone will last for 203 hours on a single battery charge. However, if it continuously scans for other peer devices, the battery is completely exhausted within 10 hours when using Bluetooth, and under 4 hours using Wi-Fi.

To mitigate this, Virtual Compass must balance the energy devoted to sensing and maintaining the social graph against the accuracy of the system. Scans that are too frequent will drain energy, and scans that are too infrequent will increase the latency for peer localization – device arrival or departure will go undetected until the next scan interval. Virtual Compass uses three techniques to reduce the number of scans without significantly degrading localization accuracy.

4.1 Adaptive Bluetooth Scanning

We observe that repeated scans are unnecessary in a static environment, such as when there are no other devices around, or when none are moving. Virtual Compass uses this observation to adapt the scan interval. Every device keeps track of changes in its neighbor graph and accordingly adjusts its scan interval – aggressively scanning the environment when the neighbor graph changes, and increasing the scan interval otherwise.

To track the change in its neighbor graph, a device calculates the number of one hop, N_1, and two hop paths, N_2, that have changed between successive scans. Figure 4 illustrates an example. From the first scan at time T_1 to the second scan at time T_2, $N_1 = 2$ one hop paths have changed, and $N_2 = 1$ two hop paths have changed. We assume applications that want to sense social interactions will care more about changes

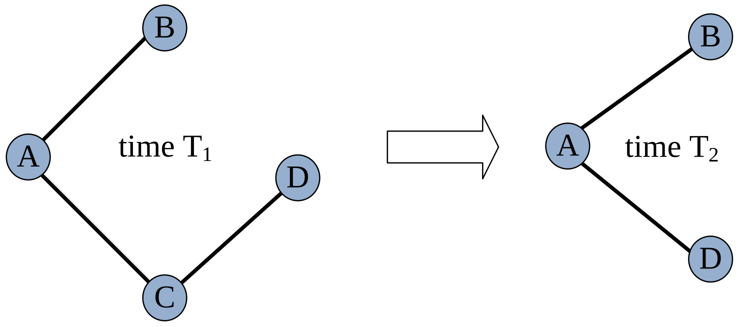

Figure 4: An example illustrating the calculation of the change metric for adapting the Bluetooth scan interval. In this example, the change metric between T_1 and T_2 is 1.9, which is high enough to trigger a reduction in the scan interval.

in the immediate vicinity of the mobile device and hence we more heavily weight changes in the one hop neighborhood. We compute a change metric as $p * N_1 + (1-p) * N_2$, where p is a constant. When this metric is less than a threshold x, we increase the inter-scan interval by 10 seconds. If the metric is above a threshold y, we halve the scan interval. We do not scan more frequently than once every 10 seconds and we do not allow the scan interval to increase beyond 10 minutes. Furthermore, we halve the scan interval every time it reaches 10 minutes. We use values of 0.9, 1 and 1 for p, x and y respectively. While these values are arbitrary and could be tied to an application, they work well in our experiments. The results of a simple experiment showing the behavior of this technique are shown in Figure 5. The scan interval additively increases until new devices are introduced or removed in the neighbor graph, at which point the scan interval is halved.

This method can be easily extended to other metrics depending on future application requirements. For instance, an application may care about sensing small changes in the distance between peers. In this case, the change metric can include distance in the calculation. An application may want to weight different peers based on their significance in the social network. We are exploring these alternatives and their usefulness to different applications.

Between two successive scans, which can be as long as 10 minutes, we leave the Bluetooth radio on. As Table 1 shows, the idle energy consumption of Bluetooth is small. Hence, keeping Bluetooth always on imposes a small energy burden on the system. Moreover, with the Bluetooth radio on, it can respond to its peer's scans and the corresponding neighborhood graph is always complete. In contrast, the idle power consumption for Wi-Fi is comparable to scanning. Therefore, the radio needs to be turned off between scans. However, this implies that adaptive scanning for Wi-Fi is infeasible—if different peers wake up at different times, their scans will result in incomplete neighbor graphs. Therefore, for Wi-Fi we periodically (every 1 minute at wall clock time) turn on the radio and put it in scan mode. Mobile phones synchronize their wall clock time with the cellular infrastructure. Even if disconnected from the cellular network, clock drift in the order of one or two seconds is not a significant issue since Wi-Fi scanning takes several seconds (see § 6).

4.2 Cloud Coordination

There are significant periods of time when a device is completely alone. Figure 6 shows how often Bluetooth scans by 150 participants [6] found other devices. On average, each mobile phone found no other Bluetooth devices 41% of the time. While it is possible that other devices were present but did not have Bluetooth

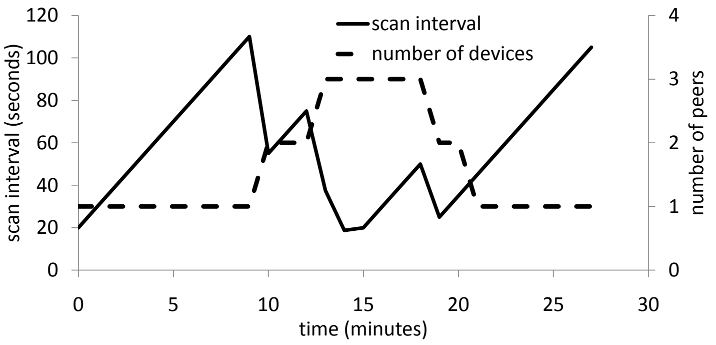

Figure 5: This graph shows the Bluetooth scan interval for a device over time. Initially, it had only one peer device in the vicinity. After 10 minutes, we introduced a second peer device, and at 12 minutes a third peer device. At 18 minutes we removed one device, and then a second one at 20 minutes.

discovery enabled, or were discoverable over the longer range of Wi-Fi, this finding fuels our belief that there are periods of time when a device is completely alone. Hence we can save energy on devices during these periods by keeping Wi-Fi off and not initiating Bluetooth scans until a new device arrives. However, the primary challenge is to detect device arrival without using Bluetooth and Wi-Fi. We observe that many mobile devices are almost always connected to the Internet via a cellular data connection such as 3G. Hence, a simple service running on the Internet can inform the device when there are other devices in the vicinity.

In Virtual Compass, each mobile device uploads its approximate geographic location to this service. This location is calculated using low-energy, coarse grained GSM localization. The list of GSM cellular towers that are in the vicinity and the RSSI values are used to compute a rough geographic location. Each time its location changes, the device updates the service. When the device believes it is alone (no neighbors in Bluetooth and Wi-Fi scans), it will periodically ask this Web service whether there are any other devices in the vicinity running Virtual Compass. If there are no peers around it, the device will keep its Wi-Fi radio off and not scan on Bluetooth. Otherwise, it adjusts its scan interval and Wi-Fi wakeup interval as described previously. Since periodic polling on a radio such as 3G consumes a considerable amount of energy, Virtual Compass uses a push-based technique to notify the device when other nodes are around. Inspired by Cell2Notify [12], a Virtual Compass device uploads a Request-for-Notification (RFN) bit to the Web service when it thinks it is alone. For each device with the RFN bit set, the Web service keeps track of other device arrivals in the vicinity of the sleeping device and will notify it, which then resumes Wi-Fi and Bluetooth scanning. We describe our implementation of this notification in the next section.

4.3 Leveraging Application Behavior

In addition to exploiting user mobility to reduce energy consumption, a cloud service allows us to also exploit application behavior. Some applications that use peer localization may not need the neighbor graph maintained all the time, even though the applications are still running. For example, an application that shows the user a map of nearby friends and how to get to them does not need the neighbor graph if the user is not interacting with the phone. Scanning in this scenario wastes energy. However, not scanning, and hence not participating in multi-hop discovery, could degrade localization accuracy for other devices where their

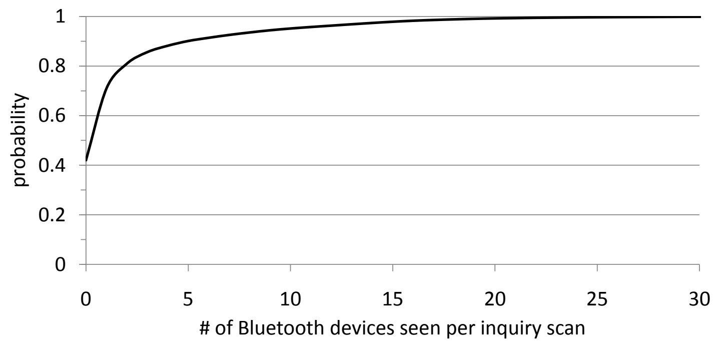

Figure 6: CDF of the number of Bluetooth devices seen in periodic scans from 150 Nokia N95 mobile phones. This data covers five months in 2008 [6]. The graphs shows that 41% of the time, on average, these devices did not discover any other Bluetooth devices.

users are actively interacting with the phone. We suspect that there are significant periods of time when every phone in the vicinity is simultaneously not in use. To detect this scenario, Virtual Compass detects when the back-light for the screen on a mobile device turns off. We then assume that the user is not using the application and upload this bit of information to the Web service along with the device's rough geographic location. When Virtual Compass polls the Web service to find out how many devices are in the vicinity or uses the notification service, it also learns how many of them have the back-light on. If no devices are actively being used, then it keeps the Wi-Fi radio off and does not scan on Bluetooth. If *any one* device in the vicinity has the back-light on, then it resumes normal discovery behavior. Unfortunately, if an application uses peer localization to log social interaction in the background, instead of displaying an interactive map, then this technique cannot be used.

5 Implementation

We have implemented Virtual Compass on the Windows Mobile 6.0 operating system that runs on a variety of mobile phones. While we have also ported Virtual Compass to the Windows Vista operating system, we focus on the mobile phone version in this section. Virtual Compass runs entirely at the application layer, and does not require modifications to the Bluetooth and Wi-Fi drivers, nor to the network stack. Our software architecture, depicted in Figure 7, consists of four main components: native radio modules, cloud services, peer localization service and applications.

Native radio modules: Virtual Compass interacts with many radios (GPS, Bluetooth, Wi-Fi, and GSM) using native APIs exposed by the Windows Mobile OS. To access the Wi-Fi radio when the device is in suspension state S3, we use a `PPN_UNATTENDED` state. This consumes slightly more energy than S3, but allows us to access the Wi-Fi radio.

For device discovery and propagating the neighbor graph, as described in § 3, Virtual Compass requires every device to periodically broadcast its ID and the IDs of and distance to each of its peers. The application layer provides the ID to be used in Virtual Compass. To broadcast this information without the additional

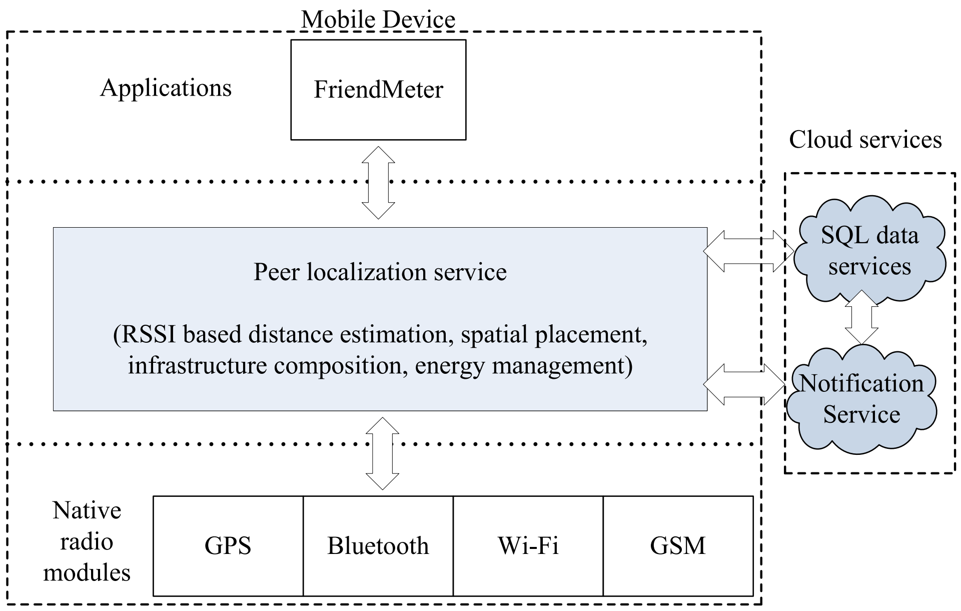

Figure 7: Software architecture of Virtual Compass

latency of explicitly forming a network, we use the Beacon-Stuffing approach [16] for Wi-Fi, and a similar technique for Bluetooth. Our beacon formats for Bluetooth and Wi-Fi are in Figure 8. For Bluetooth beacons we modify the 2048 bytes available for the device name, while for Wi-Fi beacons we embed this information in the 32 byte SSID. The small size of the Wi-Fi SSID limits the size of the neighbor graph that can be encoded in the beacon. To solve this problem, we could use two techniques proposed by Beacon-Stuffing [16]: use 256 byte IE blobs, or fragment large strings across beacons that are then reassembled at the receiver. We have not implemented either solution, and in our current implementation, we limit the neighbor graph embedded in beacons to immediate one-hop peers, thus effectively limiting peer localization to a maximum of two-hops.

One problem with using the Bluetooth radio for peer localization is that it may interfere with Bluetooth headset usage during phone conversations. A scan in the middle of a conversation will disrupt the phone call. To avoid this problem, we trap the *incoming phone call* and *phone call talking* events from the Windows Mobile OS and stop Bluetooth scanning if either event is active. We resume scanning once these events have ended.

Cloud services: Virtual Compass uses the *SQL Server Database Service* (SSDS)[5] over the Internet for coordinating Wi-Fi radio wake-ups and Bluetooth scans, as described in § 4.2. SSDS has the following components: (a) Authority: this is the top-most level of containment hierarchy under which all the data for a particular SSDS login is stored. (b) Container: an authority is a collection of containers. (c) Entities: each entity inside a container stores any number of user-defined properties and values. Virtual Compass uses a single authority, under which there is a separate container for each geographic region, under which there is a separate entity for each device. The peer localization service moves the device's entity to the appropriate container based on cellular tower IDs and RSSIs from the GSM radio and updates a bit indicating whether the

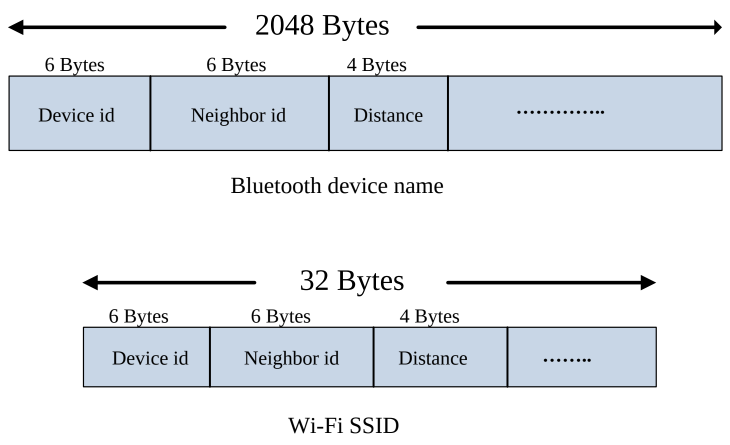

Figure 8: Format for Wi-Fi and Bluetooth beacons. The device ID and details of the neighbor graph are encoded in the beacons.

screen back-light is on. Virtual Compass can use a push (notification-based) and polling scheme to download information on neighbor positions. For polling it periodically downloads the contents of the container it is in to determine if it is alone.

When using the notification scheme, each Virtual Compass device uploads its current position based on cell tower IDs and RSSIs. When a device does not find any neighbors on a Bluetooth and Wi-Fi scan it uploads a RFN (*Request for Notification*) bit and the device's phone number to the cloud and stops scanning on Bluetooth and switches off Wi-Fi. A notification service runs on an Internet server which constantly downloads the location of all Virtual Compass devices using SSDS. It calculates whether any Virtual Compass device is near a node with its RFN bit set. If so, it uses a `Skype` client on the server to make a phone call to the device using a special caller ID number. The device traps the incoming phone call event, and if it recognizes the special caller ID number, it ends the call and resumes scanning on Bluetooth and Wi-Fi.

Peer localization service: The location service runs the distance estimation and spatial placement algorithms from § 3 to produce a 2D map of where peer devices are. The distance estimation model that we use to convert an RSSI measurement to distance and uncertainty is described in Figure 9. We derived the model using extensive measurements in several indoor environments using three different laptop brands (IBM Thinkpad, Sony Vaio, HP Pavilion) and two different HTC mobile phones (TyTNII, Touch Cruise) in an effort to be agnostic to the type of device being used. We refer the reader to prior work [42, 17] on how to build such models. The location service also implements the geographic translation of the map, as described in § 3.4. If an application enables this feature, the service will query GPS every 10 minutes, and also connect to nearby devices over Bluetooth to exchange latitude and longitude coordinates. The service also manages the Wi-Fi radio sleep and scan schedule, Bluetooth scanning interval and interfaces with the cloud services to reduce energy consumption as described in § 4. It feeds the entire map to the application layer.

Applications: We have implemented the *FriendMeter* application using Virtual Compass. FriendMeter uses Virtual Compass to sense the distances between the user and her friends who are in the vicinity.

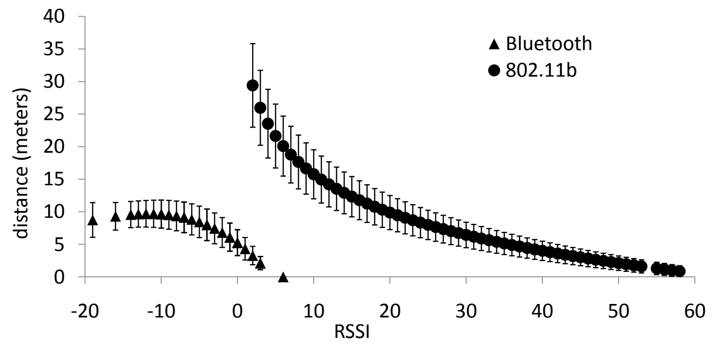

Figure 9: The figure shows the model we use in Virtual Compass to convert a Bluetooth or IEEE 802.11b RSSI measurement to distance. The corresponding uncertainty is shown using error-bars. We derived this model from extensive measurements using different laptop brands and HTC mobile phones. The RSSI values are unit-less integers reported by the drivers and it is unclear how to convert them to dBm. Virtual Compass is agnostic to the units of these values.

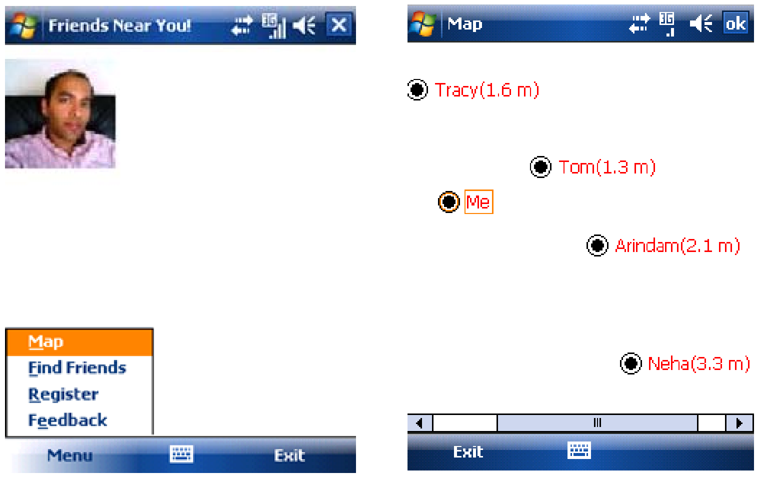

Figure 10: Screen shots of the simple UI in FriendMeter. The left image shows the start-up screen which shows the user's Facebook profile photo. The right image shows the spatial placement of nearby friends. Even though there are 5 other devices in the vicinity, the application only shows the user and 4 other users – it does not display users that are not in the friend list.

FriendMeter is designed with two purposes in mind – a short-term use and a long-term use for the sensed information. In the short-term, the results from Virtual Compass are used to show the user a map that can be used to find and meet her friends. In the long-term, the time-varying distances measured between the user and her friends can be used to infer social interactions. These inferences can be used to cluster friends in social applications, such as Facebook, based on proximity. Each friend can be metered by the amount of physical social interaction. Our implementation shows the user a map and records a history of the map, but currently does not alter their friends list.

FriendMeter uses the Facebook API to connect to Facebook, authenticate the user and get her list of friends. It uses a unique numerical Facebook login id—provided by Facebook as the mobile device's ID. This facilitates identifying the user on each peer device, but as we note in § 7, there are some privacy implications. FriendMeter displays a map of all the user's friends in the vicinity. It also displays the photographs of the nearby users and their interests, hobbies, and other information. Screen shots from the application are in Figure 10. Even though the underlying peer localization service provides a map with many devices, FriendMeter filters out those that are not in the user's friend list.

6 System Evaluation

We evaluate the performance of Virtual Compass by focusing on the following three key questions:

- How accurate are Virtual Compass's distance estimates and spatial placement?
- How much energy does Virtual Compass consume?
- How quickly does Virtual Compass adapt to changes (e.g., when a new device arrives, or one departs)?

In answering these questions, we also examine the impact of scale: how does the number of participating devices affect Virtual Compass? We present several detailed micro-benchmarks, and explore trade-offs available when setting the parameters of Virtual Compass.

6.1 Experimental Setup

We evaluate Virtual Compass on the Windows Mobile and Windows Vista operating systems. Our testbed consists of ten devices – an HTC TyTNII mobile phone, an HTC Touch Cruise mobile phone, four laptops and four desktops. All ten devices have IEEE 802.11b and Bluetooth interfaces, and are connected to the Internet via 3G cellular on the phones or Ethernet on the laptops and desktops. In most experiments, we deploy the devices in a $100m^2$ indoor office area, but we also evaluate larger areas of $900m^2$ and $2500m^2$ where indicated. Many experiments involve statically-placed nodes, but in those evaluating latency, we move a mobile phone into or out of the deployment area. When evaluating energy consumption, we measure the lifetime of the fully charged mobile phones while running Virtual Compass and leaving the GSM radio on. The two models have very similar energy consumption and hence we report only the HTC Touch Cruise numbers for brevity.

6.2 Accuracy of Localization

The primary goal of Virtual Compass is to accurately localize nearby peers. We evaluate this accuracy in two ways – (1) *error in pairwise distance* between two nodes – what is the difference between the physical distance and the distance that Virtual Compass predicts? (2) *spatial placement*: for a number of nodes, how different is the 2D placement that Virtual Compass presents from their actual placement?

Pairwise distance accuracy: Figure 11 shows how well Virtual Compass estimates the distance between two nodes as their physical distance is varied. As the graph shows, Virtual Compass comes very close to perfectly estimating distance. When Virtual Compass does deviate from the actual distance, it does so by a

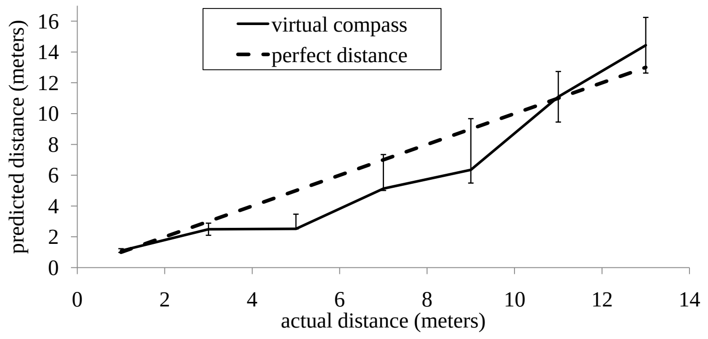

Figure 11: This graph shows the distance predicted by Virtual Compass versus the actual distance between 2 devices. In each experiment, we placed 2 devices at a measured distance. We ran multiple experiments for each actual distance, and plot the average and deviation using error bars. The $x = y$ line indicates perfect accuracy.

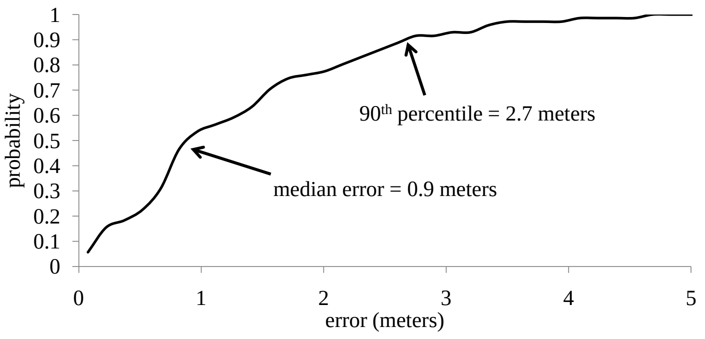

Figure 12: This graph shows the CDF of distance error (difference between actual and predicted distances). We varied the distance between 2 devices in a $100m^2$ indoor area.

System	Average Error (meters)
Bluetooth	3.40
Wi-Fi	3.91
Virtual Compass	1.41

Table 2: Nine devices were scattered in a $100m^2$ indoor area. We measured the physical distance between each pair of devices that could directly communicate and compared that to the distance reported by Virtual Compass. This table reports the average error reported by Virtual Compass, and also what the average error would have been if Virtual Compass used only Bluetooth, or only Wi-Fi.

small amount as the error bars indicate. Figure 12 shows the CDF of this distance error over a large number of placements of the two nodes. The median error is only 0.9 meters, and over 90% of the time, the error

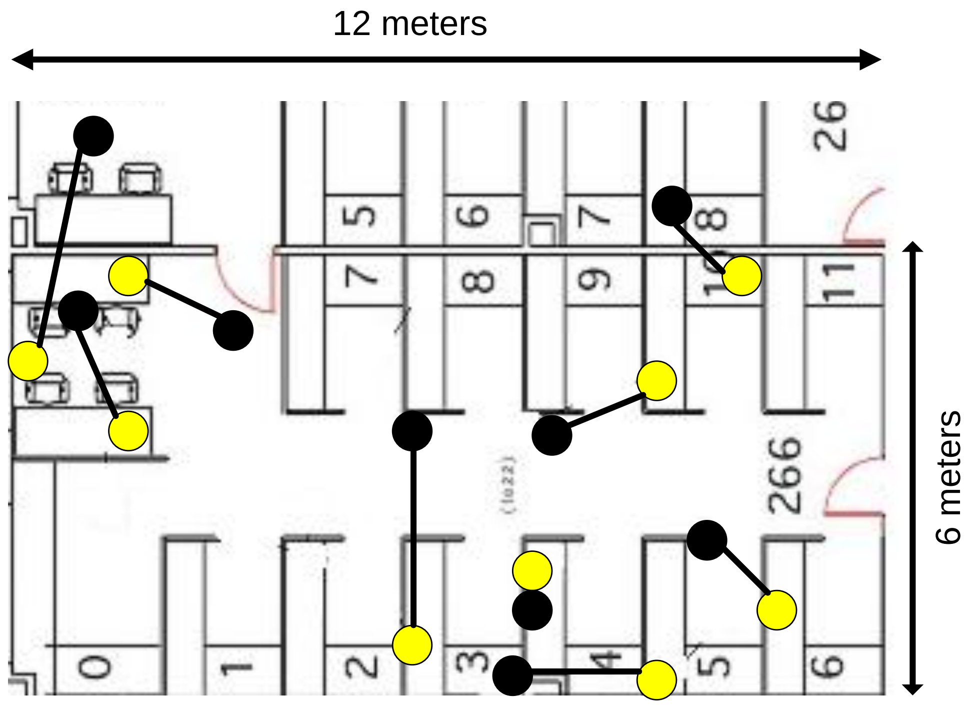

Figure 13: This figure shows a map of an indoor office environment. The light dots show the actual location of 9 devices. We overlay the spatial placement map from Virtual Compass on this figure using dark dots. A line connects each dark dot to the actual light dot that it corresponds to. The spatial placement was generated at the node on the desk between "3" and "4".

is under 2.7 meters. To examine why Virtual Compass is so accurate in pairwise distance estimation, we present Table 2, which shows the advantage of our multi-radio approach. If Virtual Compass were to use only Bluetooth radios, the average error would be quite high at 3.40 meters, or 3.91 with just Wi-Fi radios. However, by simultaneously using both Bluetooth and Wi-Fi, Virtual Compass reduces the average error to 1.41 meters (the corresponding median error is 0.9 meters).

Spatial placement accuracy: We evaluate spatial placement in Figure 13. Virtual Compass's 2D spatial placement (dark dots) almost exactly matches the actual placement (light dots) – the average distance between a light dot and the corresponding dark dot is 1.9 meters. Our accuracy is dependent on two factors : our multi-radio RSSI-based distance estimation, and our 2D spatial placement algorithm. To tease apart these two factors, we applied our 2D spatial placement to the *actual* pairwise distances between these nodes (as opposed to the RSSI-based estimates) and the average error is 0.6 meters.

The accuracy of our 2D placement algorithm also depends on the density of devices – the more devices we have, the more constraints (distance estimates) we have that allow the placement to converge faster. In Figure 14, we repeat our placement experiments while varying the number of devices, and the placement of these devices. As the number of devices is lowered, the error increases because every node is constrained by

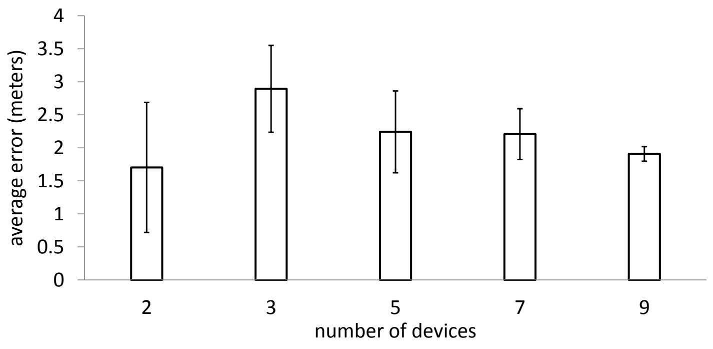

Figure 14: Using multiple experiments similar to Figure 13, we calculate the average error in 2D placement as we vary the number of devices.

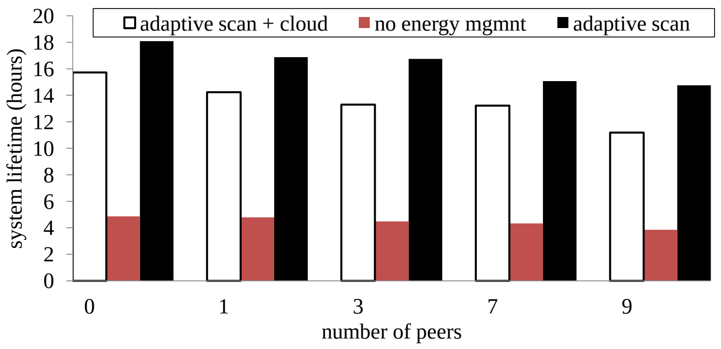

Figure 15: This graph shows the lifetime of Virtual Compass on a fully-charged mobile phone, with different energy conservation techniques and different numbers of nearby peer devices in a $100m^2$ area. In each experiment, the devices did not move. The “no energy mgmnt” bars are for runs of Virtual Compass with Wi-Fi always on and scanning every 1 minute, and Bluetooth always on and scanning every 10 seconds. The “adaptive scan” bars are for runs of Virtual Compass with Wi-Fi turning on and scanning every 1 minute and Bluetooth adaptively tuning the scan interval between 10 seconds and 10 minutes, as described in § 4.1. The “adaptive scan + cloud” bars add the polling-based cloud coordination techniques.

fewer neighbors. With just 2 nodes, the average error is small because we are effectively doing 1D placement and the error is purely a reflection of the RSSI-based distance estimation error.

6.3 Energy Consumption

While accuracy in localization is the primary goal of Virtual Compass, energy consumption is a critical concern for mobile devices. We now evaluate the benefits of the energy saving techniques from § 4. Figure 15 shows the lifetime as the number of nearby peers is varied. As we see with the “no energy mgmnt” bars, the lifetime of a mobile phone with Wi-Fi and Bluetooth always on and scanning every 1 minute and 10 seconds

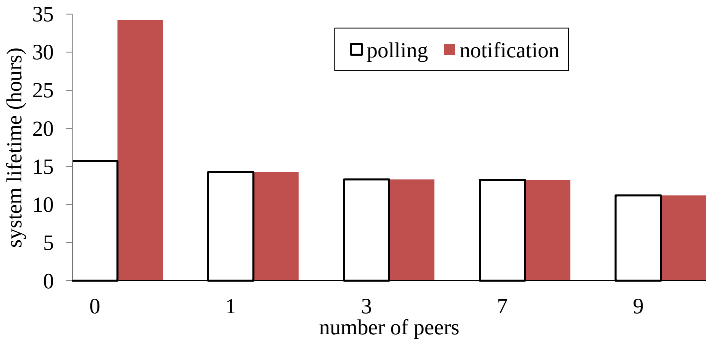

Figure 16: This graph shows the lifetime of Virtual Compass on a fully-charged mobile phone, with different numbers of nearby peer devices in a $100m^2$ area. In each experiment, the devices did not move. In the "polling" bars, Virtual Compass periodically queries the cloud service for new node arrivals, while in the "notification" bars, it uses the push-based technique for device wakeup.

back-light optimization	lifetime (hours)
off	12.07
on	15.42

Table 3: This table shows the lifetime of Virtual Compass on a fully-charged mobile phone, with the back-light optimization from § 4.3 turned off or on. We used a synthetic workload based on the Reality Mining data [8] to emulate phone usage.

respectively is dismal, at 4.8 hours with no peers and 3.8 hours with 9 peers. The slight drop in lifetime with the number of peers is because Virtual Compass has to connect over Bluetooth to every peer to get the RSSI value (this is a limitation of the Windows Mobile Bluetooth API). However, when we turn Wi-Fi on and off every 1 minute and adaptively change the Bluetooth scan interval, we see significant energy savings in the "adaptive scan" bars, from 18.0 hours with no peers to 14.8 hours with 9 peers. Even though the devices do not move, there is a drop in lifetime with the number of peers because of the Bluetooth connect issue and because with more devices, variations in the environment can temporarily appear as slight neighbor graph changes. When we include the cloud coordination scheme in the "adaptive scan + cloud" bars, the lifetime actually reduces. Periodically polling the Web service when alone (0 peers) is a significant drain on the battery. Even when not alone, our devices keep uploading their location to the Web service because of variations in the RSSI from GSM cell towers and re-association with a different GSM cell tower, despite the nodes being static in this experiment. GSM localization that is more robust to such variations should help.

In Figure 16, we show the advantage of using a notification system instead of polling. When there are no other devices around, the savings are tremendous – lifetime increases from 15.7 hours to 35 hours. Since there are no devices around, the device keeps Wi-Fi off and does not scan over Bluetooth, and does not need to poll the service over GPRS.

We now evaluate the improvement offered by the back-light optimization from § 4.3. The previous experiments do not use this optimization because we lack accurate usage models of our application. Hence in Table 3, we present an evaluation of this optimization based on emulation of the Reality Mining data [8]. The data covers a large number of users across many days and indicates when their phones are idle versus

Density (meter^2)	Lifetime (hours)	1-hop peers	2-hop peers	3-hop peers
100	11.19	9	0	0
900	11.92	5	4	0
2500	12.05	5	3	1

Table 4: This table shows the lifetime of Virtual Compass on a fully-charged mobile phone, with 9 peers nearby, across different sizes of regions. In each experiment, the devices did not move.

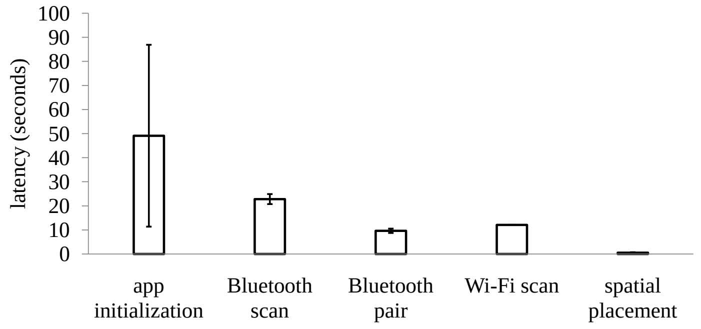

Figure 17: This graph shows the latency of various tasks in Virtual Compass, along with error bars indicating variance across several runs. A total of 10 stationary devices were placed in a $100m^2$ area. “app initialization” is dominated by communication with the Facebook Web site. “Bluetooth pair” is needed to get RSSI information from a peer.

in use. We pick 10 users at random and focus on their behavior for a random day. For periods of time when all the devices are idle, we follow our technique from § 4.3 and keep Wi-Fi off and do not scan on Bluetooth. We repeat these emulations multiple times by picking 3 different days at random, and 3 different sets of 10 users, and present average numbers in Table 3. While this emulation may not perfectly match real usage, these results show that this optimization has the potential to increase lifetime by 30%.

Finally, we present Table 4 where we evaluate the energy consumption of Virtual Compass as we vary the density of deployment. The lifetime does not significantly vary with density. There is a slight increase in lifetime as density decreases, and this is because there are fewer peers that are directly reachable over Bluetooth, and hence fewer connections need to be setup to measure RSSI.

6.4 Latency

Latency is another important metric – Virtual Compass should sense changes in the neighbor graph fast enough for applications that want to detect social interactions, and for those that provide maps in real-time to users. Figure 17 shows the overhead of different components of Virtual Compass. Bluetooth scanning is particularly slow, and we discuss this in more detail in § 7. Bluetooth pairing is needed to work around a limitation of the Bluetooth interface in Windows Mobile. The Windows Vista Bluetooth stack does pass up RSSI values from a Bluetooth scan without having to pair and connect, and so we are confident that this problem is not inherent to Bluetooth.

System	Average neighbor graph stability
Bluetooth	14%
Wi-Fi	90%
Virtual Compass	94%

Table 5: This table shows the average stability of the neighbor graph when using just Bluetooth, just Wi-Fi, or both in Virtual Compass. We placed 2 devices 10m apart, and ran each experiment for 2 hours. In the first experiment, both devices scan over Bluetooth every 10 seconds, and on average, saw the other device only 14% of the time. In the second experiment, with Wi-Fi scans every 10 seconds, each device saw the other 90% of the time. With Virtual Compass using both radios, this increases to 94%.

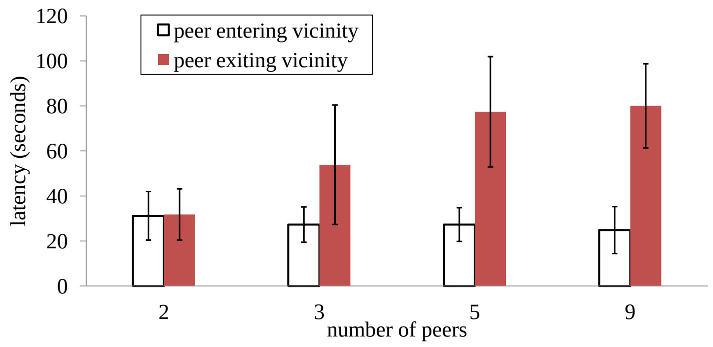

Figure 18: This graph shows the latency of Virtual Compass detecting a peer moving into or moving out of the vicinity, with different numbers of nearby, stationary peer devices. Peer exit tends to have higher latency because it is dominated by the slowest peer to remove it from its neighbor graph.

The time taken to detect the arrival of a new peer depends not only on the latency of Wi-Fi and Bluetooth scans, but also on how reliable scanning is. In Table 5, we present the probability of finding a peer device with a Bluetooth scan, Wi-Fi scan and both. Bluetooth is particularly poor because when two adjacent devices are scanning (and hence frequency-hopping) simultaneously, the probability of both being on the same channel and hence discovering each other is very low. This problem is specific to Bluetooth, as the stability of Wi-Fi is much higher. Since Virtual Compass uses both radios, it can detect the presence of a peer device more reliably than either alone.

We now evaluate how quickly Virtual Compass detects peer movement. In particular, we consider: (1) time elapsed between a peer entering the vicinity of a device and the peer showing up on the map, and (2) time elapsed between a peer leaving the vicinity and it disappearing from the map. We evaluate both latencies in Figure 18. The latency of detecting a new peer is dominated by the frequency of scanning – in steady state, Bluetooth scanning occurs once every 10 minutes, but Wi-Fi occurs every minute. Since the graph shows the average across many runs, the average latency for detecting a new peer is 30 seconds, because of Wi-Fi scanning. Peer departure can be a higher latency operation as the number of peers increases because all peers have to remove the exiting peer from their neighbor graph, else it will still appear in the map due to multi-hop discovery. Hence Bluetooth's slower scan time dominates peer departure latency.

Reducing the scan interval of Wi-Fi and Bluetooth can reduce latency, but it comes at the cost of energy. Figure 19 explores this trade-off. The second set of bars at 11.36 hours corresponds to the 5 peers bars from

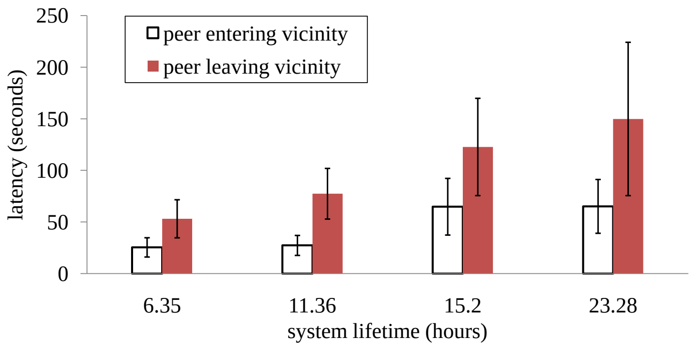

Figure 19: This graph shows the trade-off between the latency of detecting a peer moving into or moving out of the vicinity, against the lifetime of Virtual Compass on a fully-charged mobile phone. In each experiment, we placed 6 stationary devices in a $100m^2$ area and either introduced a new peer device or removed one. We varied the Wi-Fi scan interval between 30 seconds and 4 minutes to get the different sets of bars.

Figure 18. We can double the lifetime to 23.28 hours at the cost of doubling latency. However, halving the lifetime to 6.35 hours does not significantly reduce latency. Hence we believe that our choice of the Wi-Fi wake-up and scan interval of 1 minute and the Bluetooth limits of 10 seconds to 10 minutes offer the best trade-off between latency and energy consumption.

7 Discussion

We now discuss optimizations that can further improve accuracy, latency and energy consumption in Virtual Compass. We also briefly discuss some security and privacy issues.

Improving accuracy: While Virtual Compass uses a single RSSI-distance profile, we could use different profiles for different environments, such as outdoors versus indoors. This would require a mobile device to determine if it is outdoors, and then apply the corresponding RSSI-distance profile. We are exploring two ways to solve the problem of detecting that the user is outdoors. First, if a GPS signal is available, then we can assume the user is outdoors. Second, we can use user feedback.

Reducing latency: Virtual Compass's latency in detecting node movement is significantly impacted by Bluetooth scanning. Two devices that simultaneously scan over Bluetooth can miss each other because each may use a different frequency hopping sequence such that the two devices never end up on the same channel at the same time. To alleviate this problem, we are investigating certain Bluetooth 1.2 chipsets that allow us to use *enhanced inquiry* which is supposed to make discovery not only guaranteed reliable, but also finish in under 5 seconds. However, for backward compatibility with Bluetooth 1.0, we understand that most chipsets do not use this by default.

Reducing energy consumption: Not all mobile devices have similar energy budgets. A fully-charged laptop has a lot more energy than a fully-charged mobile phone. Furthermore, some environments may have powered desktops with wireless interfaces. We posit that it is beneficial for mobile phones to offload the task of aggressively scanning for device movement to nomadic infrastructure that is energy rich. The nomadic infrastructure can scan very frequently, and if it detects that a new device has come into range, or a device has moved or left, then it can signal other devices to scan and re-compute the peer map. This signaling can occur using our cloud coordination scheme. We are investigating this scheme as a more general technique to offload computation to more powerful, nomadic infrastructure.

Privacy and security: There are some privacy and security issues that we have not addressed in Virtual Compass. In our current implementation, a user's numeric Facebook ID is her mobile device's ID in peer localization. In our application, she sees only those other users that are her Facebook friends. However, our underlying peer localization component has a complete map of all devices in the vicinity. A wily user could potentially get at this information and discover the identities of strangers nearby.

Alternatively, we could use a random number for the device ID, which periodically changes. Each device would register this ID with an applet on the Facebook website. Any device that wants to discover the user identity behind a device ID will have to query the applet, which can verify if that user is a friend.

A more adversarial problem can occur if a malicious node lies about its location, or claims that certain other nodes are close to it when in fact they are not. In this way, the malicious node can obfuscate its location, or worse, add significant error to the location of other nodes.

8 Conclusion

Most of today's mobile social applications use *absolute* location to locate nearby peers, which is often difficult to obtain with reasonable accuracy in indoor environments. In this paper, we describe Virtual Compass, a peer-based localization system for mobile phones, which provides *relative* positioning by placing peers on a 2D plane without requiring any infrastructure support. Virtual Compass enables many emerging mobile applications that want the ability to sense social interactions: it provides the distance between different people which can then be combined with external information about those people's social relationships. A key area of future work is to use this information to build applications that automatically infer of social context of such interactions.

Virtual Compass leverages the multiple radios available on today's smartphones to provide the accuracy needed for the above applications It use several energy management techniques that frugally use radios without compromising location accuracy. We have implemented Virtual Compass for Windows Mobile phones. We have implemented a simple application, FriendMeter, which uses Virtual Compass to sense the distances between a user and her Facebook friends who are in the vicinity. We evaluate Virtual Compass on a nine node testbed, and our results show that it places a device with an average distance error of only 1.9 meters. Virtual Compass's energy management algorithms produce a battery lifetime that is four to seven times that of a device that does not use sophisticated energy management to provide peer localization.

References

[1] Bluehoo. `http://bluehoo.com`.
[2] Dodgeball Social Networking. `http://dodgeball.com/`.
[3] Ekahau Wi-Fi-based Real-time Tracking and Site Survey Solutions. `http://ekahau.com`.
[4] Loopt. `http://loopt.com`.

[5] Microsoft Azure SQL Data Service. `http://microsoft.com/azure/data.mspx`.
[6] Nokia Nokoscope Data. obtained via private communication.
[7] Pantopic Social Networking. `http://pantopic.com/`.
[8] Reality Mining Dataset. `http://reality.media.mit.edu/`.
[9] Rummble Social Networking. `http://rummble.com/`.
[10] Ubisense. `http://ubisense.net`.
[11] Versus Technologies. `http://versustech.com`.
[12] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless wakeups revisited: energy management for VoIP over Wi-Fi smartphones. In *MobiSys*, 2007.
[13] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based User Location and Tracking System User Location and Tracking System. In *INFOCOM*, 2000.
[14] G. Borriello, A. Liu, T. Offer, C. Palistrant, and R. Sharp. WALRUS: Wireless Acoustic Location with Room-Level Resolution Using Ultrasound. In *MobiSys*, 2005.
[15] N. Bulusu, J. Heidemann, and D. Estrin. GPS-Less Low-Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 2000.
[16] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman. Beacon-Stuffing: Wi-Fi without Associations. In *HotMobile*, 2007.
[17] R. Chandra, J. Padhye, A. Wolman, and B. Zill. A Location-based Management System for Enterprise Wireless LANs. In *NSDI*, 2007.
[18] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *SigComm*, 2004.
[19] S. Gaonkar, J. Li, R. R. Choudhary, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *MobiSys*, 2008.
[20] S. Guha, R. Murty, and E. G. Sirer. Sextant: a unified node and event localization framework using non-convex constraints. In *MobiHoc*, 2005.
[21] A. Haeberlen, E. Flannery, A. Ladd, A. Rudys, D. Wallach, and L. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *MobiCom*, 2004.
[22] M. Hazas, C. Kray, H. Gellersen, H. Agbota, G. Kortuem, and A. Krohn. A Relative Positioning System for Co-located Mobile Devices. In *MobiSys*, 2005.
[23] L. Holmquist, J. Falk, and J. Wigstrom. DOLPHIN: A Practical Approach for Implementing a fully Distributed indoor Ultrasonic Poisitioning System. In *Ubicomp*, 2004.
[24] L. E. Holmquist, J. Falk, and J. Wigström. Supporting group collaboration with inter-personal awareness devices. *Journal of Personal Technologies*, 3:13–21, 1999.
[25] A. Hopper, A. Harter, and T. Blackie. The Active Badge System. In *InterCHI*, 1993.
[26] J. Krumm and K. Hinckley. The NearMe Wireless Proximity Server. In *Ubicomp*, 2004.
[27] K. Laasonen, M. Raento, and H. Toivonen. Adaptive on-device Location Recognition. In *Pervasive*, 2004.
[28] A. M. Ladd, K. E. Bekris, A. Rudys, L. E. Kavraki, and D. S. Wallach. Robotics-based location sensing using wireless Ethernet. *Wireless Networks*, January 2005.
[29] H. Laitinen, J. Lahteenmaki, and T. Nordstrom. Database Correlation method for GSM Location. In *VTC*, 2001.
[30] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place Lab: Device Positioning using radio beacons in the Wild. In *Pervasive*, 2005.
[31] A. LaMarca and E. de Lara. Location systems: An introduction to the technology behind location awareness. *Synthesis Lectures on Mobile and Pervasive Computing*, 2008.

[32] M. Maroti, B. Kusy, G. Balogh, P. Volgyesi, A. Nadas, K. Molnar, S. Dora, and A. Ledeczi. Radio interferometric geolocation. In *SenSys*, 2005.

[33] E. Miluzzo, N. D. Lane, K. Fodor, R. A. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application. In *SenSys*, 2008.

[34] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *SenSys*, 2004.

[35] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. Beep Beep: A High Accuracy Acoustic Ranging System Using COTS Mobile Devices. In *SenSys*, 2007.

[36] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *MOBICOM*, 2000.

[37] R. Stoleru, T. He, J. A. Stankovic, and D. Luebke. High-accuracy, low-cost localization system for wireless sensor network. In *SenSys*, 2005.

[38] A. Varshavsky, E. de Lara, J. Hightower, A. LaMarca, and V. Otsason. GSM indoor localization. In *Pervasive and Mobile Computing journal*, Dec. 2007.

[39] A. Varshavsky, D. Pankratov, J. Krumm, and E. D. Lara. Calibree: Calibration-free Localization using Relative Distance Estimation. In *Pervasive*, 2008.

[40] R. Want and A. Hopper. Active badges and personal interactive computing objects. *IEEE Transactions on Consumer Electronics*, 38(1):10–20, 1992.

[41] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, 1997.

[42] G. Zanca, F. Zorzi, A. Zanella, and M. Zorzi. Experimental comparison of rssi-based localization algorithms for indoor wireless sensor networks. In *REALWSN*, 2008.

[43] P. Zhang and M. Martonosi. Locale: Collaborative localization estimation for sparse mobile sensor networks. In *IPSN*, 2008.

[44] Z. Zhong and T. He. MSP: Multi-Sequence Positioning of Wireless Sensor Nodes. In *SenSys*, 2007.