LEAP: A Low Energy Assisted GPS for Trajectory-Based Services

Heitor S. Ramos
Federal Univ. of Alagoas
Maceió, 57072-970
Brazil
heitor@ic.ufal.br

Tao Zhang
Pennsylvania State Univ.
University Park, PA 16802
USA
zhangtao@cse.psu.edu

Jie Liu
Nissanka B. Priyantha
Aman Kansal
Microsoft Research
Redmond, WA 98052
USA
{liuj,bodhip,kansal}@microsoft.com

ABSTRACT

Trajectory-based services require continuous user location sensing. GPS is the most common outdoor location sensor on mobile devices. However, the high energy consumption of GPS sensing prohibits it to be used continuously in many applications. In this paper, we propose a Low Energy Assisted Positioning (LEAP) solution that carefully partitions the GPS signal processing pipeline and shifts delay tolerant position calculations to the cloud. The GPS receiver only needs to be on for less than a second to collect the sub-millisecond level propagation delay for each satellites signal. With a reference to a nearby object, such as a cell tower, the LEAP server can infer the rest of the information necessary to perform GPS position calculation. We analyze the accuracy and energy benefit of LEAP and use real user traces to show that LEAP can save up to 80% GPS energy consumption in typical trajectory-based service scenarios.

Author Keywords

Assisted GPS, Mobile, Location

ACM Classification Keywords

H.5.m Information Interfaces and Presentations: Misc.

General Terms

Design

INTRODUCTION

GPS receivers are ubiquitous in smart phones today. Location is a dominant user context information for many mobile applications, such as traffic, navigation, search, advertising, social networking, and personal reminders. While some applications are satisfied with a single location fix — e.g. the current latitude and longitude of the device — many context-aware services require or can take advantage of continuous location traces. We call these services *trajectory-based services* (TBS). In TBS, a mobile device collects a time series of location information, and possibly sends them to a cloud server at once, to obtain services.

For example, many crowd sourcing applications for traffic [11], popular destinations [8], or cab sharing [5] require users to upload their trajectory periodically to the server for analysis. In Place-Its [14] and for location based reminders [10], the device needs to continuously monitor the user's location to deliver reminders or offers. In Predestination [7], the authors show the possibility of predicting people's destination based on a short driving trajectory. Base on such prediction, a mobile search engine can return local businesses in users' driving direction rather than those behind them. Trajectories can also help validate users' current locations by imposing physical constraints. When a person "checks in" at a location (e.g. using services like Foursquare[1]), the immediate past trajectory gives the service provider more confidence that the user is not lying about the current location.

There are several ways for a smart phone to obtain its current location, for example through the GPS receiver, WiFi or Bluetooth signatures, Cell tower ID or signal strength triangulation, etc. Among them, GPS-based positioning gives the best accuracy and best out-door coverage. However, it is well known that continuous GPS sensing is energy consuming, and that a typical smart phone will drain its battery completely in less than half a day with always-on GPS [9, 12, 19, 4]. Several techniques are proposed in the literature to reduce the overhead of continuous location sensing: 1) by using alternative sensors if the accuracy requirement is not high [9]; 2) by turning on the GPS only when significant motion is detected [12]; or 3) by combining multiple location requests from the application layer together [19]. However, in all the previous work, a GPS receiver is treated as a black box that outputs user coordinates at certain energy expenses.

In this paper, we take a deep look at the principle and processing stages of GPS receivers and propose a new operation mode for mobile GPS sensing, called *Low Energy Assisted Positioning* (LEAP). In the LEAP mode, a GPS receiver can shut down most of its internal components and

[1] http://www.foursquare.com

UbiComp'11, September 17–21, 2011, Beijing, China.

heavily duty cycle its RF front-end. Instead of decoding time stamps from satellite signals, performing Least Square location computation, and outputting the (latitude, longitude, altitude) locations, LEAP only outputs the sub-millisecond parts of the time-of-flight (ToF) of the GPS signals[2]. These sub-millisecond parts of ToF, called code phases (CP), can be detected easily without decoding any information from the satellite packets.

Using a technique known as coarse-time navigation (CTN) [16], for each set of the CPs from every visible satellite, together with a known location of a nearby object (e.g. the cell tower that the phone locks to), a server can easily compute the locations for the mobile device, or provide location/trajectory-based services. Thus, LEAP does not require any hardware change to current GSP sensors. It only needs accessing low level CP data and controlling receiver duty cycling, which we argue, should be made available by current GPS receivers.

LEAP can be classified as a type of Mobile Station Assisted GPS (MS-A AGPS or AGPS-A). However, compared to typical AGPS-A, which sends either the raw data or decoded pseudorange (full ToF) to a server, LEAP carefully manages which part of GPS sensing and computation must be done in the receiver, while others can be deferred to the cloud. Although the idea applies to any delay tolerant location calculation, such as photo tagging and single instance location-based services, the benefit is most significant for TBS when a set of CPs can be sent to the cloud at once to amortize the communication cost. Given that energy cost of communicating a packet over a 3G link is comparable to obtaining a GPS fix, LEAP is not a replacement for current GPS receivers when the user only needs to obtain a single instantaneous location coordinate locally.

We analyze the duty cycle period for LEAP mode and use real traces to evaluate accuracy and energy savings for using LEAP. We show that with 10 ms code phases, LEAP gives ± 40 m location error. With 300 ms code phases, LEAP can be as accurate as a standalone GPS. For typical TBS such as trace logging and local social networking (such as Google Latitude), LEAP can save up to 80% energy on GPS sensing than regular GPS.

The rest of the paper is organized as following. We first give a brief background on the GPS signals and GPS receiver principles. Then, we present the LEAP solution based on a coarse-time navigation technique. We examine LEAP's energy benefit by analyzing the timing and data rate at each GPS processing stage. LEAP is evaluated both for accuracy depending on receiver active duration and for energy saving using real user traces and two representative TBS scenarios.

GPS BASICS

We start with a brief (and much simplified) description of how GPS systems work. For a more formal treatment of the principles of GPS and A-GPS, please refer to [6, 16].

[2]The time it takes for the GPS signal to travel from the satellite to the GPS receiver is typically 75ms

GPS Signals

A GPS receiver obtains its location from receiving and processing digital communication signals sent from GPS satellites. There are 31 GPS satellites or space vehicles (SVs), each orbiting the earth two cycles a day. A set of ground stations monitor satellites' trajectory and health, and send the satellite parameters to the satellites. In particular, there are two kinds of trajectory information: the *almanac*, which contains the coarse orbit and status information, and the *ephemeris*, which contains the precise information of satellite trajectory. All satellites are time synchronized to within a few nanoseconds.

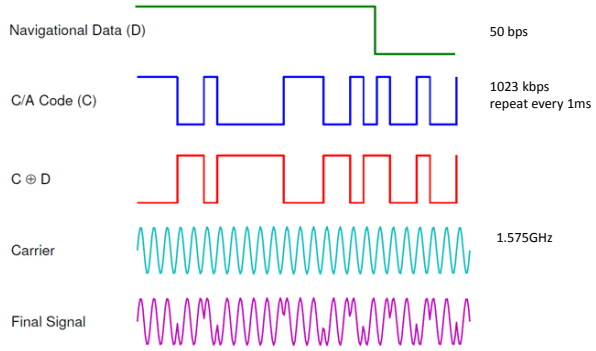

Figure 1. An illustration of GPS signal modulation scheme.

As illustrated in Figure 1, the satellites simultaneously and continuously broadcast time and orbit information through CDMA signals at 1.575 GHz towards the earth. The transmission data rate is at merely 50 bps. Apart from the data, each satellite encodes this signal (CDMA encoding) using a satellite specific C/A code of length 1023 chips at 1023 kbps. Thus, the C/A code repeats every millisecond resulting in 20 repetitions of the C/A code during each data bit sent.

A full data packet from a satellite broadcast is 30 seconds long, containing 5 six-second long frames, as shown in Figure 2. A frame has a preamble, called Telemetry Word (TLM), and a time stamp, called Handover Word (HOW). Ephemeris of the transmitting satellite and the almanac of all satellites are contained in each data packet. In other words, a precise time stamp can be decoded every 6 seconds, and the high accuracy satellite trajectory can be decoded every 30 seconds. The ephemeris information is constantly updated by the ground stations. In theory, the ephemeris data included in the SV broadcast is only valid for 30 minutes.

These data rates explain why standalone GPS may take about 30 seconds or more to obtain a location fix, since all information has to be received and decoded from the satellite signals. In mobile devices, the course-grained satellite trajectory parameters are downloaded from a server. Thus, a low accuracy time to first fix (TTFF) can be reduced to 6s.

GPS Receiving

In order to compute its location, a typical GPS receiver needs three pieces of information: 1) a time stamp [3] 2) the satel-

[3]When the time differences between the receiver and the satellites are within a few seconds, the receiver time can be another variable

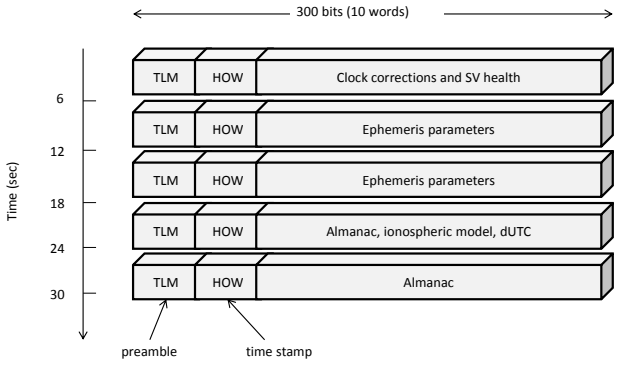

Figure 2. The frame content of a GPS packet of length 1500 bits

lites' orbits at the time, and 3) the approximated distances (called the pseudorange) from each satellite to the receiver at the time. Among those, the key is to obtain the pseudoranges, which are computed from the time of flight of the RF signals from each satellite to the receiver.

The RF signals travel 64 to 89 milliseconds from a satellite to earth surface. Notice that light travels 300 km/ms. So in order to obtain an accurate position, the receiver must track time to the microsecond level. The millisecond part (NMS) and the sub-millisecond (subMS) parts of the propagation time are detected very differently. While NMS is decoded from the packet frames, the subMS propagation time is detected at the C/A code level using correlations.

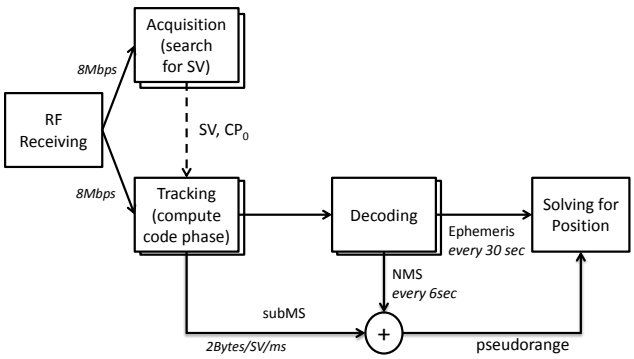

Figure 3. A simplified illustration of GPS processing stages

Figure 3 shows a simplified diagram of different stages of GPS signal processing and location calculation. When the GPS receiver is just turned on, it has no information on what SVs are visible and their code phases [4].

- The *Acquisition* stage is run when the GPS receiver starts up. The goal of the Acquisition phase is to start receiving the data transmitted by the SVs visible to the GPS receiver by correctly decoding the received signal. Acquisition phase also measures the Code Phase values as a byproduct. To decode the data from a given satellite, three unknowns have to be estimated.

in Least Square optimization.

[4] in A-GPS, the visible SVs are included in the assisted information, but the receiver still need to acquire the satellites to lock to them.

First, it is necessary to determine if the given satellite is visible to the receiver. The presence of a given satellite can be determined by detecting the presence of its C/A code in the received GPS signal. Second, although the satellites transmissions are centered around a 1.575GHz *carrier* frequency, the signals from different satellites received at the GPS receiver deviate from this due to the Doppler frequency shift caused by the relative motion between the satellites and the receiver. It is necessary to determine these Doppler shifts to decode the data from a given satellite. Third, since the satellite signal is encoded by a 1023 bit C/A code, the received signal has to be decoded by multiplying it with the C/A code of the corresponding satellite at the correct time instance (CDMA decodings). Although the C/A codes are well known, the exact timing of when the signals should be multiplied is unknown and it depends on the receiver position. Since the C/A codes repeats every 1ms this unknown fractional ms time represents the Code Phase of the corresponding satellite.

If the receiver has no knowledge of the current SV arrangement in the sky and the precise time, it needs to search through all possible C/A codes, Doppler frequency shifts, and code phases. In typical GPS hardware, it can takes 30 seconds to finish acquisition. This is a relatively expensive operation, so A-GPS helps this process by providing the visible SV information off line. However, it is still necessary to complete the frequency and code search for each SV in view.

- Once the satellite signals are acquired, the receiver enters a relatively inexpensive *Tracking* stage, which keeps feedback loops to adjust phase locks and delay locks and maintain the code phases in the receiver in sync with those from the satellites. In the continuous mode, the tracking loop runs every millisecond.

- With correct tracking, the receiver can *decode* the packets sent by the SVs. In general, without assistance information, the receiver needs to decode SV ephemeris every 30 minutes (its valid time span) and time stamps every 6 seconds. Decoding is energy consuming since it has to run tracking continuously for the packet duration in order to receive all the bits. With A-GPS, the receiver is not required to decode ephemeris, but it must still decode HOW.

- Given ephemeris and propagation delays obtained from code phase and HOW, the GPS receiver performs *position calculation* using constraint optimization techniques, such as Least Square minimization. This is usually done on the phone's main processor. With receivers' latitude, longitude, altitude, and precise time, the receiver needs a minimum of 4 SVs in view.

It is important to understand the data rates at each stage of processing. While baseband C/A code has a bit rate of 1.023 Mbps, the signals are typically oversampled at 8 MHz to achieve better code phase estimation. A code phase (subMS) can be computed from any 1 ms of the baseband signal. Once computed, it requires only 2 bytes per satellite (10-bit Code Phase and 6-bit satellite ID) to represent it. On the other

hand, it takes 6 or 30 seconds to decode packet contents from the signals. During which time, the tracking circuits of the receiver cannot be turned off. Thus, as we will realize in the next section, code phases are the most efficient information to represent the device location when other information can be obtained separately.

Energy Analysis

The energy consumption of typical GPS sensing consists of two parts — (1) the GPS receiver that needs to acquire, track, and decode the signals sent from the GPS satellites, and (2) the main processor that controls the receiver and performs position calculation.

As a concrete example, consider the energy use for a GPS receiver on an HTC HD2 smartphone. We placed the mobile phone in a sleep state and ran a background service to use only the GPS receiver and obtain a location fix. The power draw during a location sensing period is shown in Figure 4. In this mode, aside from the nominal power usage of the phone for maintaining network presence (7.3mW in the figure), the only other energy used is that for GPS based location sensing, consisting of the GPS receiver chip-set, the phone's main processor, and the minimal supplemental circuitry required to power the two former components. Ignoring the power bursts during activating and de-activating the system, the power draw during location sensing is 387mW. This is consistent with other measurements of phone based GPS energy consumption performed on other devices. To place this number in perspective, a typical phone battery capacity of 1000mAh will allow such a GPS receiver to operate continuously for only 9.5 hours, assuming that the phone is not used for anything else.

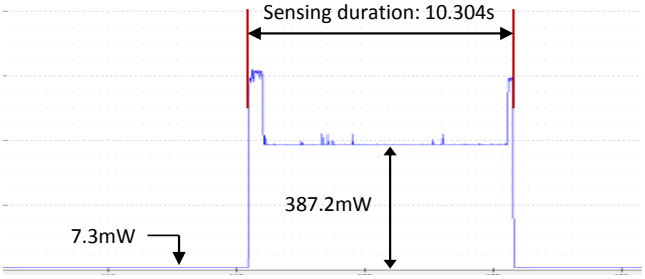

Figure 4. Power trace showing phone energy use for GPS location sensing.

LEAP SOLUTION

The fundamental idea of LEAP is to use coarse-time navigation (CTN) [16] techniques to trade off between local GPS signal processing and communicating sufficient data to the cloud. CTN, also called instant GPS, is a known GPS technique, which have unfortunately received little attention. One reason for this is that standalone GPS does not have a communication link to the cloud. Another reason is that CTN does not necessarily improve TTFF, which has so far been the primary optimization goal for mobile GPS design. However, it provides the best foundation for partitioning GPS receiving stages between the device and the cloud.

Coarse-Time Navigation

Coarse-time navigation does not require decoding any data from satellite packets. Instead, the GPS receiver only needs to compute the code phase (CP) for each visible SV and have a reference to a known "nearby" location. The key insight is that since light travels at 300 km/ms, two locations close enough to each other (e.g. within 150km) will observe the same NMS, modulo the boundary conditions that can be disambiguated by snapping the CP to the nearest integer.

GPS location estimation involves 4 unknowns–the (x, y, z) coordinates of the GPS receiver and the precise time. Solving for these 4 unknowns requires a minimum of 4 visible satellites. However, CTN adds another source of error, called *coarse time error*, due the lack of a precise time reference on the receiver side. Correcting this error requires an additional visible satellite, making it necessary to have 5 visible satellites for computing location. We use the coarse time error model described in [16] to correct this error.

Even though the millisecond part of the pseudorange can be estimated by the initial position and the coarse-time, any small error on this estimate can lead to a very inaccurate position calculation. For instance, an error of 1 ms produces results about 300 km away from the actual location. Integer rollover is the main source of such errors during the estimation process. Van Diggelen [16] proposes the following method to reconstruct the full pseudoranges avoiding the integer rollover problem. The travel time can be written as

$$NMS^{(k)} + \varphi^{(k)} = \tau^{(k)} - \delta_t^{(k)} + b + \epsilon^{(k)} \quad (1)$$

$$= \hat{\tau}^{(k)} - d^{(k)} - \delta_t^{(k)} + b + \epsilon^{(k)} \quad (2)$$

where $NMS^{(k)}$ and $\varphi^{(k)}$ are the millisecond and sub-millisecond components of the transmission delay to the satellite k, respectively, $\tau^{(k)}$ is the actual travel time corresponding to the satellite k, $\delta_t^{(k)}$ is the satellite clock errors obtained from the ephemeris at the a priori coarse time for the satellite k, b is the common bias, and $\epsilon^{(k)}$ represents some unknown errors (tropospheric model error and other stochastic errors).

As $\tau^{(k)}$ is unknown, it can be written as $\tau^{(k)} = \hat{\tau}^{(k)} - d^{(k)}$, where $\hat{\tau}^{(k)}$ is the estimated travel time from the a priori position at the coarse time of transmission, and $d^{(k)}$ is the error in $\hat{\tau}^{(k)}$. The method involves the selection of a reference satellite, $k = 0$, where $NMS^{(0)} = round(\hat{\tau}^{(0)} - \varphi^{(0)})$ is the millisecond part of the pseudorange of the reference SV[5]. This value is used to reconstruct the millisecond travel time of all other satellites relative to the reference satellite. Thus, if we subtract the Eq. (2) from the reference satellite full travel time we get

$$NMS^{(k)} = NMS^{(0)} + \varphi^{(0)} - \varphi^{(k)} + \left(\hat{\tau}^{(k)} - d^{(k)} - \delta_t^{(k)} + b + \epsilon^{(k)}\right) - \left(\hat{\tau}^{(0)} - d^{(0)} - \delta_t^{(0)} + b + \epsilon^{(0)}\right) \quad (3)$$

[5][16] recommends the use of the highest satellite in view as reference, and provides a good reason for that

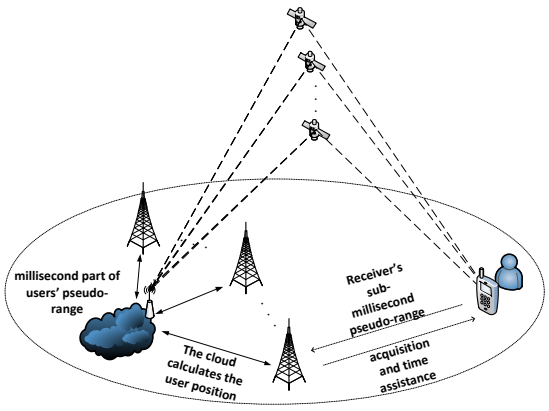

Figure 5. LEAP system overview

We still do not know the values of $d^{(0)}$ and $d^{(k)}$, but considering that we have an initial position and coarse time close to the correct values (about 100 km and 1 min), the order of magnitude of $(-d^{(k)} + \epsilon^{(k)} + d^{(0)} - \epsilon^{(0)})$ is less than about 150 km, thus, we can correctly estimate $NMS^{(k)}$ by

$$NMS^{(k)} = round\left(NMS^{(0)} + \varphi^{(0)} - \varphi^{(k)} + (\hat{\tau}^{(k)} - \delta_t^{(k)}) - (\hat{\tau}^{(0)} - \delta_t^{(0)})\right). \quad (4)$$

In summary, the full pseudorange is estimated by using NMS as the millisecond part, and the sub-millisecond is obtained by the code phase estimate. CTN can be computed either on the device or on a back-end server. In order to compute it on the device, the phone needs to periodically update the ephemeris data every 30 minutes, and have a database of the precise locations of a set of landmarks. Alternatively, since the only device dependent data are the code phases for each visible SV, the location computation can be moved completely to the cloud.

LEAP for Trajectory-Based Services

LEAP is a realization of the CTN principle to reduce the energy expense for obtaining trajectory-based services. By not decoding GPS signal and by offloading location calculation to the cloud, LEAP can aggressively duty cycle GPS receivers for energy efficiency.

Mobile devices have two advantages when using the CTN principle. First, they have a fairly good time synchronization due to the requirement from the cellular network. So it is very unlikely to have more than 2 seconds of time error. Secondly, the devices are almost always associated to a cell tower, if there is one within range. Notice that the communication range of a cell tower is typically less than 5 km, which makes the cell towers ideal reference locations for applying CTN.

Figure 5 shows the system overview of the LEAP approach.

- **On the device:** The mobile device receives necessary assistance for SV acquisition phase and has a coarse time synchronized with the cellular system (as in typical AGPS). Whenever a location reading is requested from the application, the device performs a SV acquisition and a few milliseconds of tracking to estimate the code phase for each SV. It stores a *LEAP tuple*, including the code phases, the current time stamp, and the associated cell ID, and then shuts down the receiver. Note that a GPS receiver can see at most 12 SVs at the same time. So each LEAP tuple is at most 40 bytes long, including two bytes for each code phase, 8 bytes for time stamp, and 8 bytes for cell tower ID. When a TBS is requested based on T seconds of history, the device sends the LEAP tuples collected in the last T seconds as part of the service request. A traditional mobile GPS would send three double precision numbers for latitude, longitude, and altitude (total of 24 bytes) and an 8-byte time stamp for each point in a trajectory. So the communication cost between LEAP and traditional mobile GPS is comparable.

- **On the server:** To support the LEAP mode, each TBS server first converts the set of LEAP tuples into a trajectory. This is relatively straightforward using Eq. (4) when the server maintains ephemeris for all SVs and a (rough) location database for all cell towers. Since the locations are computed on the server, mechanisms for improving GPS accuracy, such as Wide Area Augmentation System (WAAS) [1] and differential GPS can be easily applied.

  In US, National Geodetic Survey computes the orbits of GPS satellites and publishes them using the Standard GPS (SP3) Format. The ultra rapid data has a 6-hour latency, which can be extrapolated to estimate real-time orbit information at relatively high accuracy. Obviously, a set of servers can receive and decode the same ephemeris data sent from the satellite for most up to date orbit information.

  While carriers typically do not publish their cell tower locations, there are open source projects, such as OpenCellID[6], that build cell tower databases by crowd sourcing. Note that LEAP does not require precise cell tower locations, since they only provide a rough reference point to compute NMS.

In comparison, a naive method of mobile-station-assisted A-GPS can send raw GPS data, with the amount of 1023 bits/ms to a server. As we will show later, to obtain accurate location comparable to standalone GPS, one may need more than a hundred milliseconds of raw data. Thus, LEAP leverages local computation that GPS receivers already do to reduce the amount of data being sent.

Receiver Duty Cycling

LEAP's energy benefit comes first and foremost from offloading packet decoding and location calculation to the cloud. However, if the RF frontend and the satellite tracking process have to run continuously between consecutive location samples during trajectory collection, then a significant part of the GPS receiver have to remain active all the time.

[6] http://opencellid.org

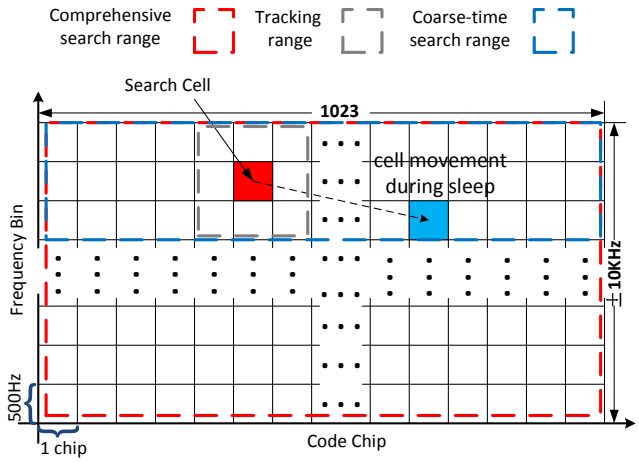

Figure 6. GPS acquisition search range

On the other hand, shutting down and resuming only the satellite tracking component is not as trivial as it seems. The time gap between two tracking processes can cause feedback loop instability, introduce large CP errors, or even lead to a failure to acquire any satellite. So how to duty cycle a GPS receiver must be carefully considered. In this session, we present a mechanism for fast re-acquisition based on previous tracking results, so that the receiver can be turned off completely between readings for better energy saving.

It is necessary to first describe the typical acquisition process when there is no prior knowledge of time or code phases. As shown in Figure. 6, when a GPS receiver starts, it must search in both frequency bins (caused by Doppler effect) and code phase bins (caused by imprecise time) to lock each SV. Given ± 10 KHz frequency uncertainty, 500 Hz frequency bin and code length of $1,023$ chips[7], in the worst case there are $41,943$ search cells in total (the dashed area in Figure. 6). Without any acceleration, the search input data rate is 1 ms/cell. Thus a brute-force search process will take more than 40 sec. exhaustively traverse the search space. Much work has been done to speed up the initial acquisition (e.g. massive parallel correlation, parallel frequency search [17], parallel code phase search [18]), however, little has been done on search space reduction based on a previously known SV lock — a process we call *reacquisition* or *reacq* for short.

Assume that the GPS receiver has locked to each visible SV, computed their CPs, and went to the sleep mode for T seconds. When it wakes up, the local carrier frequency and code phase would have changed due to the movement of satellites, of the user, and the local clock bias. The changes of frequency and code phase imply cell movement in Figure. 6. Based on the previously locked cell, and the duration of sleep, the goal is to reduce search space for LEAP to a small neighborhood, rather than searching the whole space again.

The key observation we want to leverage is that a small moving distance of the user in a short period of time only causes small frequency and code phase changes. For example, for less than a 3 km distance, all points fall within ± 3 Hz Doppler frequency shifts and ± 11 chips code phase shifts. Table. 1 sets some parameters for deriving the reacquisition search space. We assume the user moves at a maximum speed of 160 km/h and 8 SVs are visible. We also assume the GPS receiver has 12 correlation channels, which is very conservative comparing to the state-of-art devices.

[7]We simplify the problem description here. In practice, the search unit of code phase is usually half chip, which means totally $2,046$ search cells for one frequency bin.

Table 1. Coarse-time Search Settings (courtesy from [16])

# of Satellites	8
# of Correlation Channels	12
Satellite Velocity	± 3 chips/s(at ± 800 m/s)
GPS Receiver Frequency Drift	3 ppm
Maximum User Speed	160 km/h
Doppler Effect by User Motion	± 468 Hz
Reference Frequency Error	± 157.5 Hz(± 100 ppb)
Vertical Error	± 100 m
Frequency Bin	500 Hz
Code Length	$1,023$ chips
One Tracking Iteration	1 ms

At the maximum 160 km/h($\approx$44.5 m/s) speed, T-second sleep duration results in $\pm 44.5T$ m horizontal location difference. If $44.5T < 3$km , the Doppler shift from the reference position (last position) is less than 3 Hz. In addition, a T-second sleep introduces extra frequency error up to T s$\times 0.8$ Hz/s = $0.8T$ Hz. By adding these to the local frequency drift and Doppler shift caused by user motion, the maximum assisted-frequency search space is $\pm(157.5 + 468 + 3 + 0.8T)$ = $\pm(628.5+0.8T)$ Hz. For example if $T = 60s$ we can reduce the frequency search space to within -676.5 to $+676.5$ Hz of the previous lock. Considering that each frequency bin is 500 Hz, searching in the nearby 3 bins is sufficient.

Next, the reacq process needs to search all possible code phases, at a minimum rate of 1 s for each frequency bin. If we assume 12 correlation channels searching in parallel, reacq can finish searching the first satellite within 250 ms.

After the first satellite is locked, reacq can use the code phase differences between SVs from previous tracking results to further reduce the code phase search range. Considering ± 11 chips code phase shift due to user motion, ± 11 chips code phase shift due to local clock frequency uncertainty, and ± 3 chips per second code phase shift due to satellite motion, the reference clock error can be estimated as $2 \times 11 + 2 \times 3 \times T$ chips. For example, for $T = 10$ s, we need to search ± 82 chips, while for $T = 60$ s, we need to search ± 382 chips.

Given that bin searching is a relatively simple operation in hardware, reacq can be done in real time. So the time consumed by the receiver is the time it needs to collect the data. Putting everything together when $T = 10$ s, the total reacquisition time is ≤ 414 ms; and for $T = 60$ s, it is $\leq 1,014$ ms.

Based on the analysis above, LEAP uses a three phase duty

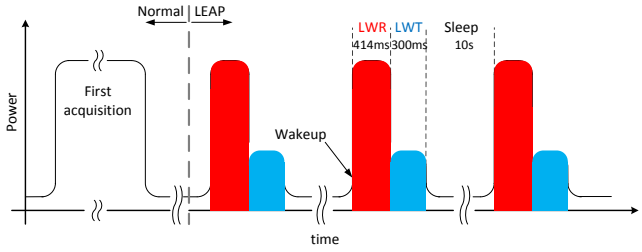

Figure 7. Duty cycling of the GSP receiver in LEAP.

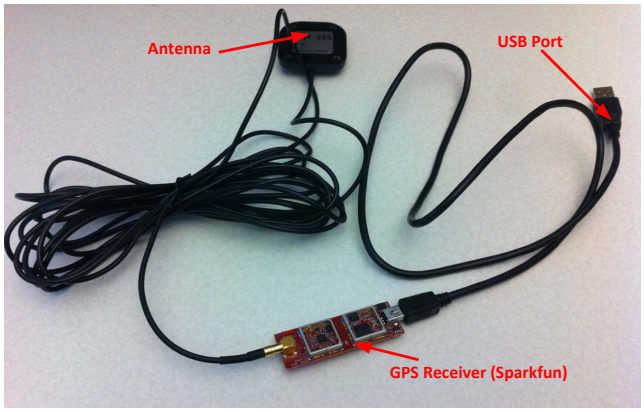

Figure 8. The GSP data collector used for experiments

cycle after the initial full acquisition — sleep, reacquisition, and tracking — as shown in Figure. 7. Assuming maximal vehicle speeds, the longest time that the GPS receiver can sleep is about a minute. Beyond that the small variation assumption is violated and the device must perform an exhaustive search again. After sleeping, LEAP performs a light-weight reacquisition to re-initiate the tracking loop. The length of the tracking loop is determined by the accuracy requirement of the application. One set of code phases is produced by each tracking loop. Due to the noise in the extremely weak GPS signal and the multi-path effect, the receiver must collect sufficient number of CP samples to tease out location errors. In the next section, we will show empirically that 300 ms tracking loops give good enough location outputs.

EVALUATION

Since no off-the-shelf GPS chip outputs low-level code phase information through a programmable interface, we evaluate LEAP using raw GSP data traces and a software-defined GPS implementation [3], which we modified to simulate receiver duty cycles and computation offloading.

We collect about 10 sets of raw GPS data at three different locations using a SiGe GN3S sampler version 2 dongle (available from Sparkfun Electronics[8]), as shown in Figure 8. Each data set includes the baseband GPS signal sampled at 16 MHz for at least 30 second long to cover complete GPS packets.

[8] http://www.sparkfun.com/commerce/product_info.php?products_id=8238

We use a standalone Garmin GPS Map 60CS[9] device to survey the "ground truth" of these locations, which themselves have about 15 m accuracy. A fourth location is used to simulate a nearby cell tower. During our experiments, the Gamin GPS produces dilution of precision (DOP) values 5 or 6, which are consider good or moderate. Table 2 shows all the positions (latitude and longitude), the precision, and the number of satellites in view at the moment we collected the positions.

For ephemeris data, we used Ultra-Rapid Orbits available on the International GNSS Service (IGS) website[10]. These orbits provide precision estimates of the SV's position and clock rate. Thus, it provides all information required by LEAP.

To evaluate the accuracy of our approach, we calculate the errors between LEAP's output and the outputs from the unmodified software implementing standard GPS processing. It is noteworthy that the values presented here represents how far our approach is from a similar standalone GPS implementation when the same signal trace is used. In addition to regular GPS error sources, LEAP adds the following possible errors: (i) the position is calculated by using code phases samples closer to each other than regular GPS; these samples are more likely to suffer temporary noise, (ii) the LEAP lock loops (PLL and DLL) run for less time than typical GPS; sometimes it may incur to less accurate results, and (iii) the use of CTN technique can also cause numerical errors.

We conduct our experiments with the following LEAP examples: (i) LEAP010, (ii) LEAP050, (iii) LEAP100, (iv) LEAP300, and (v) LEAP500. The numeral suffix corresponds to the amount of milliseconds of the signal used in the tracking phase – 10, 50, 100, and 300 milliseconds.

Given one-millisecond of signal, we are able to estimate a pseudorange for each SV, and therefore the receiver's position. Thus, we compute the final position using two different measures of central tendency: (i) the mean, and (ii) the median of a given number of chunks. Taking average from a set of positions is a common strategy in regular GPS to remove noise. However, median is also known to be resilient to outliers. In our analysis, we consider the maximum of 50 equally spaced chunks used for noise reduction. That is, even when we run 500 tracking loops, a set of code phases are only collected every 10 ms.

The number of satellites

Figure 9 shows the average absolute error with respect to the number of satellites in view, for each location and strategy used (mean or median). It shows that the the number of satellites in view poses a strong influence on the error quality. Observe that, generally, when the receiver can see only five satellites (the minimum for LEAP), all strategies present the largest errors. When the number of satellites increases, the error decreases steeply. We can also see that median values

[9] http://www8.garmin.com/products/gpsmap60cs

[10] http://igscb.jpl.nasa.gov/

Table 2. Evaluation Data Points

Location	Latitude	Longitude	Precision (m)	# of Satellites	Distance to the "cell tower"
Assumed Cell Tower	47.64168°	−122.14081°	±5.18	9	–
Location A	47.63006°	−122.14433°	±4.57	10	1.23 km
Location B	47.64337°	−122.13928°	±4.27	10	0.220 km
Location C	47.66941°	−122.14373°	±4.88	8	3.09 km

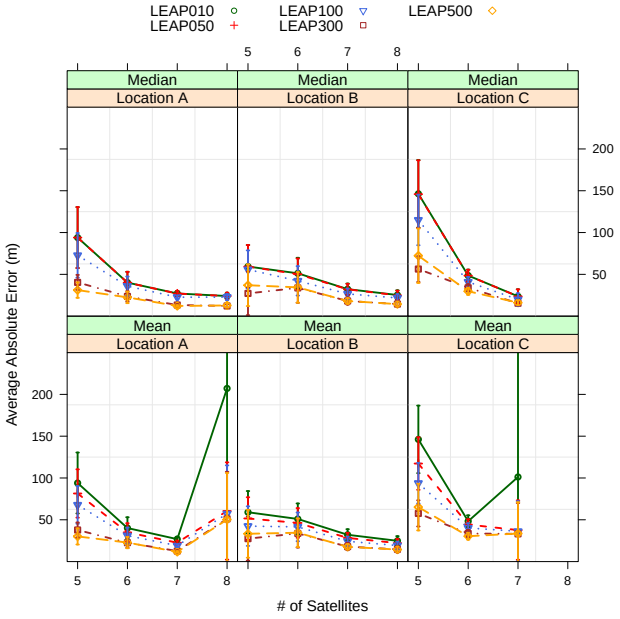

Figure 9. Error in function of the number of satellites and location

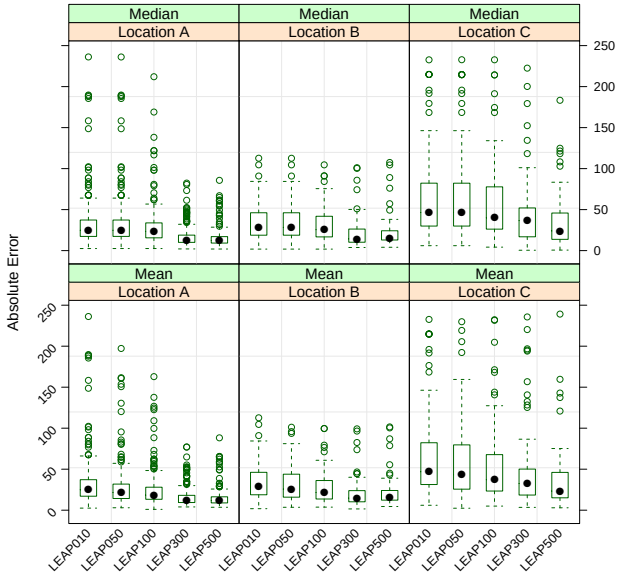

Figure 10. Error in function of the number of chunks and location. About 2% of data were considered outlier and are omited from this plot

Table 3. Outliers Statistics

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
253.9	314.6	414.8	486.6	517.8	1773.0

are more robust to outliers than the mean values, especially for Locations A and C when LEAP10 is used.

Tracking Time

Figure 10 shows the behavior of the absolute error with respect to tracking time, at various locations and using different strategies (mean or median). In the plot, the lower and upper box limits represent the 1^{st} and 3^{th} quartiles, respectively, the whiskers represent a range 1.5 times the IQR (interquartile range), and the points that eventually appear below or above the whiskers range are considered outliers. The black dots represent the mean value. To better visualize this plot, we removed the most significant outliers with error larger than 250 m. These result in 76 samples among 4000 computed. The noise and distortion in GPS signals are likely to lead to these bad points, which are not uncommon in standard GPS. Some statistics of the outliers are presented in Table 3. These bad conditioned samples can be easily detected and removed in a TBS, for instance, comparing the actual position to prior positions and checking its plausibility.

Observe that LEAP300 leads to absolute error less than 50 m for 75% (above the 3^{th} quartile) of the samples, and the mean values are typically less than 40 m. Moreover, LEAP300 only presents 4 samples into the set of removed outliers, while, for instance, LEAP10 contributes 23 samples. Those results indicate that LEAP300 represents a good trade-off between accuracy, stability, and energy saving. It is noteworthy that the median can help to avoid some outliers and should be used.

So LEAP naturally provides accuracy/energy tradeoff. If the application requires only a city block accuracy, even 10 ms tracking is acceptable. By using 300 and 500 ms, and when the median is used, the error is very similar to regular GPS devices even when the number of satellites in view is low.

Energy Consumption

The energy savings offered by using the LEAP mode in practical usage scenarios are evaluated next using real user location traces. The location traces are collected by asking volunteers to run a data collection application on their mobile devices. This application continuously samples location, stores them locally, and uploads the trace once a day to our server. The trace data set consists of 18 unique users from US, Europe, and Africa, with 1 to 20 days of data per user. Each user's trace has different periods of satellite visi-

bility depending on where they live, work, and travel.

Consider two types of scenarios:

Trajectory Tracking: For this scenario, the user location is tracked continuously throughout the day at a few minutes granularity (once a minute in our runs) at moderate accuracy (40 m). The data is uploaded when the user has wired power, so communication energy is ignored. Such usage is representative to applications that mine user behavior over time, or collect movement patterns over large populations (e.g. skyhook). The location data is not required instantaneously.

Local Social: In this scenarios, location is tracked continuously every few seconds (6 s in our runs) at high accuracy and uploaded to a cloud service every 5 minutes. Such usage represents social networking applications to find friends meeting at a public place, advertisement applications to track user movement and suggest currently relevant local services and/or products, and services that require recent motion trajectory. In this type of scenarios, the communication energy of uploading location is important.

It is tempting to compare LEAP with WiFi-based location services. If we consider proper duty cycling, the amortized energy expense is about the same for the two approaches without hardware optimization for LEAP. And both require communication to the server for location resolution. However, not all places have WiFi coverage. In our dataset, on average 25% locations do not have WiFi in sight, especially when driving on highways. In addition, WiFi services typically provide $30 \sim 100$m location accuracy, while LEAP can provide better accuracy $10 \sim 40$m with same energy expense. So we focusing on comparing LEAP with standard GPS solutions in this section.

We perform the evaluation based on trace-driven simulation. We assume that location tracking is turned off when the device does not have GPS visibility, such as indoors, and an automated mechanism using a low energy sensor, accelerometer [19], is available to resume GPS use when user movement is detected. This implies that savings from LEAP accrue only when GPS satellite visibility is available, making our comparison disadvantageous for LEAP since savings for other parts of the day are suppressed.

Energy Model: There is no GPS receiver on the market that is optimized for LEAP. For example, in the ideal case, when the GPS receiver is not involved in decoding the satellite packets, corresponding parts should be turned off. To evaluate energy benefits, we take a loose assumption that LEAP uses the same power as a regular GPS receiver when it is active. We assume the same GPS hardware for both LEAP and without LEAP, using an average draw of 400 mW when powered on (brief impulses of higher power are observed for non-LEAP GPS for certain calculations such as least squares phase, and are ignored, making the comparison disadvantageous for LEAP). So the main energy differences come from duty cycling: with LEAP, the GPS only needs to be turned on for 1024 ms for each location sample at 60 s interval and 40 m accuracy. Without LEAP, the GPS requires 5 s for each location sample, every 60 s. When sampling every 6 s with high accuracy, LEAP mode requires 414 ms per sample, while a regular GPS have to stay powered on.

To make the simulation evaluation close to reality, we also measured the energy use for accelerometer sampling (259 mJ for 1s of sampling) for motion detection, and location trace upload (10.1 J for 5 minutes of location trace) relevant to the second scenario.

Figure 11 shows the energy savings provided by LEAP. Since different users have different number of days worth of data and the absolute energy usage varies widely, we normalize the data to the energy usage on non-LEAP mode and plot only the percentage savings. Savings are significant for many users. The users for which savings are small are mostly ones for whom GPS visibility was available only for small time windows during the entire day and the energy use is dominated primarily by the accelerometer sampling. The actual energy use is very different at 60 s sampling and 6 s sampling with upload but the percentage savings are similar. This happens because the communication energy of an upload every 5 minutes is negligible compared to GPS sampling every 6 s.

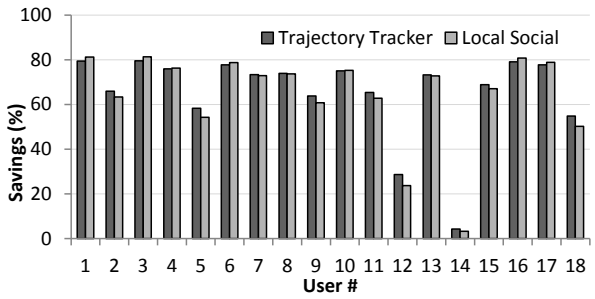

Figure 11. Energy savings from LEAP mode in two representative scenarios. Users #12 and #14 experience less energy saving than others mainly because they have little or almost no GPS visibility during the experimental periods.

RELATED WORK

GPS embraces a rich body of research by itself. Most of them around the receiver design and the signal processing algorithms in them to improve the speed and accuracy of obtaining the first fix. For instance, [15] present a comprehensive survey on signal processing techniques used in AGPS systems. Sirilo [2, 13] and van Diggelen [16] show different approaches on how to model the GPS positioning equations and how to solve them without decode any portion of data when some assistance is provided. Those solutions are useful when the receiver can use assisted data from the network, thus, they can decrease the TTFF and allow the use of GPS devices even when data decoding is not possible such as in presence of noise, shadow and incomplete data. Their work directly motivated us to study the duty cycle and energy efficiency of GPS sensing and lead us to the LEAP solution.

Energy efficiency of location sensing on mobile devices have drawn great attention in recent years. Several pieces of work look into using alternative sensors to infer when to invoke GPS sensing. For example, [9] propose a smart mechanism, namely A-Loc, which automatically determines the dynamic accuracy requirement for mobile search applications. A-Loc chooses the sensor that saves energy while attend the application's accuracy requirements. This approach relies on the idea that location applications do not always need the highest available accuracy. To do so, they use a Bayesian estimation framework to model the location and sensor errors, and thus, choose the most adequate sensor.

Paek et al. [12] propose a rate adaptive GPS-based positioning for smart-phones. They argue that, due the so-called urban canyons problem, even GPS sensor is not able to provide a high accuracy position, thus, the device should be turned off. In those situations, alternatively, they use other location mechanism such as accelerometer, space-time history, cell tower blacklisting and Bluetooth to decrease the active time of GPS devices.

Zhuang et al. [19] present an adaptive location-sensing framework to reduce the use of GPS device in various scenarios. Their framework is based on four design principles: (i) substitution, where alternative location-sensing mechanisms are used instead of GPS device; (ii) suppression, where the information provided by other sensor can suppress the use of the GPS device, for instance, if the accelerometer can identify that the user is not moving, the GPS can be turned off; (iii) piggybacking, that synchronizes location request from different applications, and (iv) adaptation that adjust sensing parameters when the battery level is low.

In comparison, we take a very different, and orthogonal, approach to solving energy efficient location sensing problem by looking into the GPS sensing principle and investigating what can be duty cycled and what can be moved to the cloud. LEAP can be used in combination with other location sensing frameworks transparently.

CONCLUSIONS

This paper presents a low energy GPS sensing solution called LEAP that is particularly suitable for enabling trajectory-based services for mobile devices. By looking into the GPS signal processing stages and applying the coarse-time navigation method, we found a sweet spot to divide GPS sensing between mobile devices and the cloud. As a result, the GPS receiver only needs to perform code phase detection for hundreds of milliseconds and thus can be aggressively duty cycled for energy saving. Through energy analysis and trace-driven simulation, we observe that the energy benefit of LEAP compared to traditional GPS on mobile phones can be up to 80%.

REFERENCES

1. F. A. Administration. Navigation services - wide area augmentation system (waas). `http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/waas/`.
2. D. Akopian and J. Syrjarinne. A fast positioning method without navigation data decoding for assisted gps receivers. *IEEE Transactions on Veicular Technology*, 58(8):4640–4645, 2009.
3. K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*. Birkhäuser, 2006.
4. S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: sharing and querying content through mobile phones and social participation. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 174–186, New York, NY, USA, 2008. ACM.
5. M. Haridasan, I. Mohomed, D. Terry, C. A. Thekkath, and L. Zhang. Startrack next generation: a scalable infrastructure for track-based applications. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
6. E. D. Kaplan and C. J. Hegarty. *Understanding GPS: Principles and Applications*. Artech House, second edition, 2005.
7. J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *The 18th International Conference on Ubiquitous Computing (UbiComp 2006)*, pages 243–260, 2006.
8. N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, and A. T. Campbell. Urban sensing systems: opportunistic or participatory? In *Proceedings of the 9th workshop on Mobile computing systems and applications*, HotMobile '08, pages 11–16, New York, NY, USA, 2008. ACM.
9. K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 285–298, New York, NY, USA, 2010. ACM.
10. P. J. Ludford, D. Frankowski, K. Reily, K. Wilms, and L. Terveen. Because i carry my cell phone anyway: functional location-based reminder applications. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 889–898, New York, NY, USA, 2006. ACM.
11. P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: using mobile smartphones for rich monitoring of road and traffic conditions. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 357–358, New York, NY, USA, 2008. ACM.
12. J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 299–314, New York, NY, USA, 2010. ACM.
13. N. Sirola. A method for GPS positioning without current navigation data. Master's thesis, Department of Electrical Engineering. Tampere Univeristy of Technology., 2001.
14. T. Sohn, K. A. Li, G. Lee, I. E. Smith, J. Scott, and W. G. Griswold. Place-its: A study of location-based reminders on mobile phones. In *Ubiquitous Computing/Handheld and Ubiquitous Computing*, pages 232–250, 2005.
15. G. Sun, J. Chen, W. Guo, and K. J. R. Liu. Signal processing techniques in network-aided positioning. *IEEE Signal Processing Magazine*, pages 12–23, 2005.
16. F. van Diggelen. *A-GPS: Assisted GPS, GNSS, and SBAS*. Artech House, 2009.
17. D. Van Nee and A. Coenen. New fast gps code-acquisition technique using fft. *Electronics Letters*, 27(2):158 –160, jan. 1991.
18. Y. Zheng. A software-based frequency domain parallel acquisition algorithm for gps signal. In *ASID'10: Proceedings of the internatial conference on Anti-Counterfeiting Security and Identification in Communication*, pages 298 –301, july 2010.
19. Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 315–330, New York, NY, USA, 2010. ACM.