Cooperative Transit Tracking using Smart-phones*

Arvind Thiagarajan arvindt@csail.mit.edu MIT CSAIL
James Biagioni jbiagi1@uic.edu
Tomas Gerlich tgerli2@uic.edu
Jakob Eriksson jakob@uic.edu
University of Illinois at Chicago

Abstract

Real-time transit tracking is gaining popularity as a means for transit agencies to improve the rider experience. However, many transit agencies lack either the funding or initiative to provide such tracking services. In this paper, we describe a crowd-sourced alternative to official transit tracking, which we call cooperative transit tracking.

Participating users install an application on their smartphone. With the help of built-in sensors, such as GPS, WiFi, and accelerometer, the application automatically detects when the user is riding in a transit vehicle. On these occasions (and only these), it sends periodic, anonymized, location updates to a central tracking server.

Our technical contributions include (a) an accelerometer-based activity classification algorithm for determining whether or not the user is riding in a vehicle, (b) a memory and time-efficient route matching algorithm for determining whether the user is in a bus vs. another vehicle, (c) a method for tracking underground vehicles, and an evaluation of the above on real-world data.

By simulating the Chicago transit network, we find that the proposed system would shorten expected wait times by 2 minutes with only 5% of transit riders using the system. At a 20% penetration level, the mean wait time is reduced from 9 to 3 minutes.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Experimentation, Performance

*This material is based upon work supported by the National Science Foundation under Grants CNS-1017877, CNS-0931550, and DGE-0549489.

SenSys'10, November 3–5, 2010, Zurich, Switzerland.

Keywords

Public transit, public transportation, bus, subway, real-time tracking, activity classification, smartphone, crowd-sourcing, power management

1 Introduction

Real-time bus tracking, where available, has been well received by transit riders. Knowing where a bus or train is at present and when it will arrive at a particular stop cuts down on waiting time, increasing efficiency while improving safety and comfort. However, many transit agencies do not yet provide tracking capabilities, due to resource constraints, red tape or lack of incentive. Also, the cost of a transit tracking deployment can be prohibitive, sometimes running into tens of millions of dollars [4, 1].

In this paper, we present a grassroots solution to transit tracking, as an alternative or complement to official systems. Rather than install and maintain an official tracking device in each vehicle, our system enables users to collectively track transit vehicles by reporting their location while inside them.

In the envisioned system, users run an application on their smartphone to learn about the location or predicted arrival time of a transit vehicle. The application remains as a background process after the user is finished with it, waiting to see if the the user eventually enters a transit vehicle. Once in a transit vehicle, the phone anonymously uploads its coordinates, contributing tracking data to a central server.

A fully automatic system requiring no manual data input is the most attractive solution. This is a hard problem that poses several technical challenges. First, knowing that the user is in a vehicle requires us to accurately distinguish between walking, stationary use and vehicular movement, *without using power-hungry and sometimes unavailable GPS*. Second, determining if the vehicle is a transit vehicle, and which one, can be challenging due to GPS error in "urban canyons" and similarities between bus routes. Non-transit vehicles such as cars operate on the same major arteries as buses, and we need to avoid misclassifying these as buses. Finally, tracking subways that operate underground is difficult because neither GPS nor WiFi/cellular localization techniques work well there.

To this end, we design several novel algorithms, which, together with a comprehensive evaluation, constitute the main contributions of this paper.

Accelerometer-based activity classification to detect

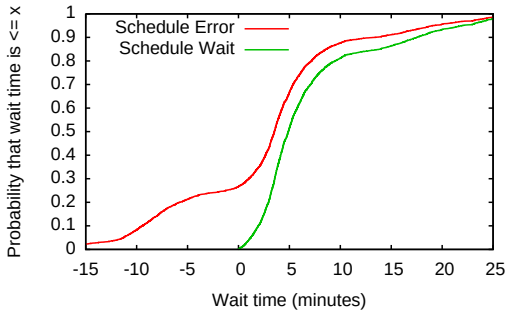

Figure 1. Measured difference between scheduled and actual arrival times of buses in Chicago.

when a user is traveling in a vehicle.

Spatio-temporal trajectory matching to determine if the detected vehicle is a transit vehicle, and which route.

Underground vehicle tracking to enable tracking of subways in addition to buses and trains.

We evaluate all components of the system on real-world data, using a combination of manually collected traces from smartphones and publicly available data from the Chicago Transit Authority (CTA). We describe smartphone-based cooperative transit tracking system, and evaluate the effect of the percentage of instrumented transit riders (penetration level) on our system through trace-driven simulation. We find that with a 5% penetration level, we can service over a third of all real-time bus tracking requests in the city of Chicago. At a 20% penetration level, over 83% of queries are serviced with real-time tracking, and the mean wait time at a bus stop is reduced from 9 to 3 minutes. These results reflect buses, and all hours of the day. For trains, and for the rush hour commute periods, results would improve further.

2 Motivation and Background

In this section, we show that real-time transit tracking information has significant benefits in terms of expected commuter wait time, and study the main technical challenges in cooperative transit tracking.

2.1 The Utility of Real-Time Tracking

Due to the dynamic nature of the road network and transit ridership, transit vehicles seldom adhere perfectly to their published schedules. Using the Chicago Transit Authority (CTA) buses as a case study, we recorded 6,300 bus-hours of real-time location traces[1], and performed a trace-based simulation to determine the accuracy of the published schedules.

Repeatedly choosing a random bus route, stop, direction and time, we looked up the next departure in the schedule, and compared this to the actual next departure, as provided by the real-time location trace.

We use two metrics to evaluate the quality of an arrival time prediction method. Figure 1 shows the CDF of these two metrics for the static schedule method: looking up the next arrival in the bus schedule. One metric, which we call "error", is the difference between the predicted next arrival time and the actual next arrival time. A negative value denotes that an arrival occurred before the predicted time. The

[1]The CTA has official real-time bus tracking devices installed in all vehicles.

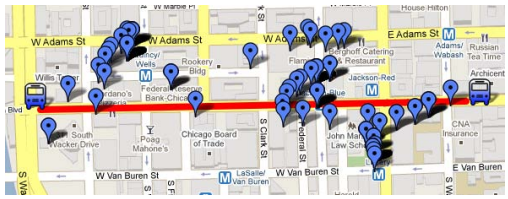

Figure 2. GPS trace and an actual trajectory of a bus ride downtown Chicago

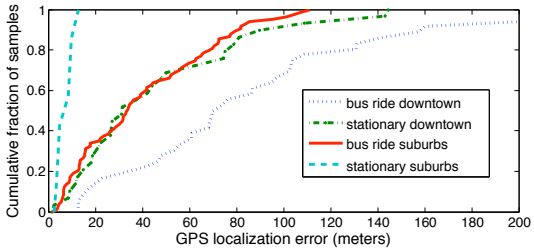

Figure 3. CDF of GPS localization errors for downtown and suburban environments

other metric, which we call "wait", assumes that the user arrives at the bus stop at the predicted (scheduled) time, and is the difference between the next arrival *after* the predicted time, and the predicted time. These values are always positive, although sometimes a negative error can result in a very large wait (consider missing the last bus of the day).

A user that arrives at the bus stop at the scheduled time would miss 27% of departures that arrive early, and encounter a mean wait time of 5 minutes (median 7 mins), with 10% of wait times being more than 17 minutes. In less than 3% of cases is the wait shorter than 1 minute.

In §4 we show that real-time transit tracking reduces the 90th percentile of the wait time by half, even by checking the predictor a single time. If the user checks the predictor repeatedly as the arrival draws nearer in time, even better gains can be expected.

2.2 Challenges

GPS Errors. The accuracy of GPS localization depends heavily on the immediate surroundings of the receiver. High-accuracy localization requires a clear line-of-sight to 4 satellites in the sky above [12], and multi-path effects also contribute to localization error [6, 12]. Both of these issues are prevalent in areas with tall buildings, or "urban canyons". Figure 2 illustrates this problem, depicting a bus ride in downtown Chicago with iPhone GPS readings as dotted markers and the actual bus route as a thick red line.

Official transit tracking systems in urban areas address this problem by augmenting error-prone GPS localization with inertial navigation systems, including gyroscopes, odometers, turn sensors and accelerometers, and use a route map to "snap" the returned location to a sensible location [4].

As cooperative transit tracking is built on smartphones, GPS errors impact our ability to accurately distinguish vehicles and estimate arrival times. To determine the magnitude of the problem, we performed GPS measurements in downtown Chicago and its suburbs for users riding in a bus and stationary 'on-street' users. We developed an iPhone appli-

cation that continuously records GPS data, and lets the user carefully mark their actual location on the map. Figure 3 shows the distribution of GPS errors as a CDF, summarized in Table 1 below.

Environment	Median (m)	90th pctile. (m)
Stationary in suburbs	6.6	12
Stationary downtown	31	99
Bus ride in suburbs	34	82
Bus ride downtown	70	160

Table 1. Measured GPS errors in different environments

The significantly higher error for bus rides vs. stationary use was due to an unpredictable reporting delay incurred by the iPhone OS when moving rapidly: GPS points from suburban bus rides were typically well aligned with the road, but poorly located *along* the road. Fortunately, this type of error has little impact on route matching performance. Nevertheless, with a median error of 31 meters and 90th percentile error of 99 meters downtown, GPS error is a significant challenge for any smartphone-based transit tracking system. §3.2 describes our system for addressing this problem, and §4.2 evaluates its performance.

Smartphone Accelerometry. Modern smartphones are almost universally equipped with built-in 3-axis MEMS accelerometers, which are cheap, power efficient, and sufficiently accurate for their primary purpose: to detect the orientation of the device for user interface and entertainment purposes. We use accelerometry to reduce our reliance on precise positioning technologies, in part to save energy when not in a transit vehicle, and in part to support underground transit tracking, where GPS and other localization technologies perform poorly.

We place no restrictions on the orientation or placement of the phone, and all our algorithms use the L2 norm of acceleration along all three axes to avoid any orientation dependence. The unknown location (pocket, hand, backpack, etc.) of the phone also means that accelerometer signatures may vary significantly, both in absolute values, and in the frequency domain.

Battery Life. Continuously using the GPS is well known to drain the battery on a smartphone quickly [16, 27]. In contrast, the accelerometer is much less power hungry, as indicated by the measurements in [16]. Here, a method is provided for using the accelerometer to distinguish between two states of mobility: stationary and moving. Our approach to using the accelerometer is similar (§3.1), but differs in that we also distinguish walking from vehicular movement. We can thereby limit GPS use to vehicle tracking.

Table 2 captures the measured battery lifetime when continuously using the different sensors available on an iPhone. Because it was not possible to turn off the phone's screen and at the same time keep the sensors running, all the experiments were run with the screen brightness set to a minimum while completely discharging the phone's battery. The results were averaged over 5 different phones. The experiments were run with the phone's screen turned on and we would expect the battery lifetime to be significantly longer if it was possible to turn the screen off.

Sensors used	iPhone 3G	iPhone 4
No sensors	18.6 hr (1.7)	16.6 hr (0.8)
Accelerometer 1Hz	19.5 hr (1.4)	17.3 hr (1.8)
Accel. 20Hz with FFT	18.3 hr (3.1)	16.9 hr (0.8)
GPS	6.1 hr (0.6)	10.1 hr (0.3)

Table 2. Mean (std.dev) battery duration, in hours, for different sensor settings.

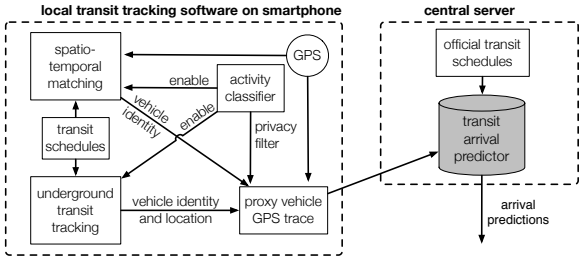

Figure 4. Cooperative transit tracking system.

The battery duration was similar for running the accelerometer with sampling frequency of 1Hz, the sampling frequency of 20Hz (with an FFT computed every 128 samples), and for discharging the battery with no sensors turned on. Between the three non-GPS scenarios we observed very little variance, with the battery lifetime around 18 and 17 hours for iPhone 3G and iPhone 4 respectively. This clearly illustrates the minor effect accelerometry has on energy consumption. With GPS enabled, battery lifetime was reduced by 42-66%. In both non-GPS and GPS cases, we would expect battery lifetimes to be significantly longer with the screen turned off. One may hypothesize that a simple exponential mean calculated at 1 Hz would result in longer battery life than an FFT on 20 Hz accelerometry, were the screen's energy consumption not drowning out the difference.

3 Cooperative Transit Tracking

Current transit tracking solutions are built on a combination of GPS localization and some form of wide-area connectivity. While real-time vehicle tracking solutions are widely available, they can be quite expensive to deploy. Many transit agencies do not have this capability today, or if they do, do not make this data available to the public.

To assist the riders of these transit networks, we propose a system based on participatory sensing using GPS-enabled mobile phones. Besides the obvious cost advantage (no additional equipment, subscription or maintenance charges are needed), our proposed system also holds a significant practical advantage: it lets us track transit vehicles without first securing the cooperation of local transit agencies, something which can be extremely difficult to achieve at large scale.

Our cooperative transit tracking system is built from several interacting components, as illustrated in Figure 4. The majority of the computation is performed on the smart-phone itself, safeguarding user privacy while conserving central server resources. The activity classifier (§3.1) monitors the user's activities and automatically determines when they are riding a vehicle. This triggers either the spatio-temporal route matching engine (§3.2), to determine whether the user is on a transit vehicle, and if so which one, or the under-

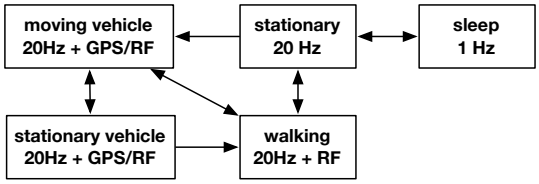

Figure 5. Finite state machine of the activity classifier. Sensors used vary by state. 1/20 Hz refers to accelerometry, RF refers to WiFi and cellular localization.

ground transit tracker (§3.3), which tracks vehicles where GPS is unavailable. The end result is a "proxy" transit GPS trace delivered to a central server, identifying the vehicle and its current location, if (and only if) the user is currently onboard a transit vehicle. The central server aggregates the produced real-time transit traces, and responds to arrival time queries based on these traces.

Arrival time prediction accuracy is a function of the most recently observed location(s) of the transit vehicle(s), which in turn depends on the total number of active users in the system. Performance evaluations of each component, and of the end-to-end system, are provided in §4. Below, we describe each component in more detail.

3.1 Activity Classification by Accelerometer

The basic activity detection algorithm takes accelerometer and GPS data as input, and determines whether or not the user is currently in a vehicle. Known techniques [17, 21, 29] include using a threshold on the user's GPS-derived velocity. This is insufficient for our application; continuously running the GPS drains the battery, buses may travel at walking speeds during heavy congestion, and GPS may be unavailable or lack sufficient precision for reliable classification.

Accurate mobility detection, supporting both indoor (underground) and outdoor mobility, is best provided by a combination of GPS (for computing velocity), WiFi localization (for detecting location changes where GPS is unavailable), and accelerometry (for detecting movements directly). However, constantly sampling these sensors will quickly drain the batteries of even the latest generation of smartphones. Thus, our goal is to minimize the use of power-hungry sensors while preserving system accuracy.

To address the problem of knowing when to use which sensor, we define a finite state machine representing the various user states we are interested in, see Figure 5. Transitions between states are limited by what is physically possible, and each state utilizes one or more sensors to detect future transitions. For example, when in the sleep state, it suffices to run the accelerometer at 1 Hz until movement is detected. When walking, sampling the accelerometer at 20 Hz lets us catch any vehicular mobility, and RF localization (cellular and WiFi) provide sufficient accuracy to determine the entry-point to a potential underground station. Once we detect vehicular mobility, we turn the GPS on to provide accurate tracking results, and turn it off after walking is detected once more. This system helps us save energy while maintaining high accuracy when the user is traveling.

The state transitions in Figure 5 depend either on the low-power motion detector (§3.1.1), the walking detector (§3.1.2) or the vehicular motion detector (§3.1.3). Underground transportation warrants special attention. Whenever a localization method is active, the most recent location is stored for later. Should vehicular movement be detected while no GPS or low-error RF localization signal is available, the most recent location is matched against underground transit stations to reveal the station where the trip started. Underground transit tracking is discussed in §3.3.

Related work: The benefits of using low-energy sensors to trigger more power-hungry sensors are well known [22, 26, 28, 15, 11]. Related work on activity detection includes [20, 18, 19, 10, 8, 23, 24]. These use a combination of accelerometry and other sensors to classify activities such as standing, walking, or riding in an elevator. In contrast with our work, this body of work uses accelerometers with known locations/orientations, and additional sensors such as audio or barometric pressure sensors which improve the classification results.

3.1.1 Low Power Motion Detection

Smartphones are likely to spend a significant fraction of their time stationary, during which time they cannot produce transit tracking data. Our system includes a simple low-power detector for possible transitions away from stationary use. It can be thought of as a wake-up mechanism for the more sophisticated detector described in §3.1.2.

Our low power motion detector samples the accelerometer at 1 Hz, and continuously computes an exponentially weighted mean and standard deviation of the X,Y and Z accelerometer readings. If an incoming sample falls outside of three standard deviations on any axis, it reports "motion detected". If the phone is static, the readings are more or less constant and lie within this band. Occasional false alarms have a negligible effect, as the 20 Hz detector described below will quickly detect that no movement is taking place, and return to the stationary state and its low-power detector. [16] describes a similar energy conservation technique.

3.1.2 Walking detection

Walking detection based on accelerometry has been studied before, though under different circumstances. In [9], the authors describe a step counter based on peaks in the accelerometry. In other work, the orientation and placement of the device were carefully controlled, providing highly accurate acceleration data. For example, in an application for medical rehabilitation, [7] used accelerometers placed at multiple locations and with a specific orientation. Another example of a carefully placed accelerometer sensor is the Nike+iPod system [5] where the accelerometer is located in a fixed place in the shoe. Most closely related to the activity classification aspect of our work is [25], which combines GPS speed with a DFT-based features to build an activity-classification decision tree. We compare against part of their algorithm in the evaluation section.

The number of significant environmental factors, such as location, walking style, physical user characteristics, clothing choice [9], etc. makes estimating the location of the device on the body from accelerometer data impractical. Recall that our envisioned users are running our software on their mobile phone, which they may be using for other activities, or may choose to stow it in a pocket, bag or purse. Thus,

we have no control over or knowledge of the orientation or placement of the device.

Our walking detector uses a technique similar to that described in [25]. Raw accelerometer values, sampled at a moderate 20 Hz, are first made orientation-independent by computing the L2-norm (or magnitude) $|a_{\{x,y,z\}}|$ of the accelerometer readings. For a sliding window **w**, we then compute its discrete Fourier transform (DFT)

$$M_k = \sum_{n=0}^{|w|-1} m_n e^{-\frac{2\pi i}{|w|}kn} \quad (1)$$

where $k = 0, \ldots, |N| - 1$.

In [25], the magnitude of the DFT coefficients in frequency bands common to walking (1-3 Hz) are used as features for classifying a walking activity. To improve accuracy we introduce an additional feature: peak frequency power. This feature is independent of the speed of walking, and captures some of the cases where the fundamental frequency (of walking) is not the peak frequency, due to placement-dependent jiggling or bouncing effects. In the evaluation section, we demonstrate greatly improved accuracy over the pure DFT and total power features used in [25].

3.1.3 Vehicular Motion Detector

Detecting vehicular mobility by accelerometer serves two purposes: (a) as an energy conserving mechanism for triggering GPS localization only when in a vehicle, and (b) as input to our underground transit tracking mechanism (§3.3).

Using the accelerometer as input, we estimate the probability that vehicular mobility is in progress. This algorithm expects accelerometer input from periods of stationary use, or vehicular movement. Our highly accurate walking detector is used to filter out periods of walking.

Based on observations from training data, we model the two distributions of acceleration samples in the moving and stationary state as Laplace distributions, with probability density function

$$f(x|\mu, b) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}},$$

where μ is the median sample of the training set, and $b = \frac{1}{N}\sum_{i=1}^{N} |x_i - \mu|$.

Given these probability density functions, we use Bayes' theorem to compute the probability that a sample x came from the moving distribution

$$p(mov|x) = \frac{f(x|\mu_{mov}, b_{mov})p(mov)}{f(x|\mu_{mov}, b_{mov}) + f(x|\mu_{sta}, b_{sta})}.$$

This probability is used as input to the subway tracking mechanism in §3.3, and to determine when to turn on the GPS, as illustrated in Fig. 5. For the latter, an HMM with two states (stopped, moving) is used to smooth out short-term variations.

3.2 Spatio-temporal Trajectory Matching

After the activity classification algorithm detects vehicular travel, the system needs to determine whether the vehicle is a transit vehicle, and if so which one, before commencing periodic location reports to the central server. This problem is technically challenging for the following reasons:

Localization Errors. Data from GPS and WiFi/cellular localization has significant errors and gaps, especially in downtown urban canyons with lots of tall buildings. Downtown is precisely where many bus trips originate or end.

Overlapping Bus Routes. A number of bus routes in downtown tend to overlap or coincide. This requires the algorithm to be careful to not mistake one bus for the other, and to be able to distinguish the routes when sufficient location information is available.

Non-transit vehicles. We do not want to require the end user to explicitly notify the application when they're on a bus. Rather, we want our algorithm to be able to distinguish bus rides from car drives automatically, and as quickly as possible. This can be challenging, as cars drive along the same major arteries as bus routes. Although cars are typically much faster and do not adhere to the bus schedule, this is not universally true – e.g., in rush hour or heavy traffic, cars and buses can be equally slow, both not adhering to the schedule. Neither can we distinguish on the basis of stop locations alone. Buses do not always stop at all bus stops on a route, and cars may stop at lights or stop signs close to major bus stops.

Hardware footprint. We want our matching algorithm to be simple to implement, and use as little phone memory and storage as possible, so as to minimize the initial download.

We use a two-stage algorithm to solve the real-time transit matching problem while addressing the above challenges. The first step is an online least squares minimization algorithm with an *ordering constraint* that finds the bus shape which is the "closest fit" for a sequence of GPS points. The second step is an intelligent *post-processing step* that distinguishes non-transit vehicles (cars) from buses, and identifies almost identical or overlapping bus routes.

Our algorithm does not rely on storing a road map. It instead matches locations directly to bus schedules, for which we use a compact, compressed representation. This results in a 5x footprint reduction (from 10 Megabytes to 2 Megabytes in our implementation), which is important in the context of a smartphone application – it can quickly download the relevant "map tile" for whichever transit network is most relevant to the user's location.

Our post-processing step uses three features to quickly disambiguate bus trips from non-transit vehicles on the same path: adherence to the bus schedule, stopping at bus stops, and *inter-stop distance*, which exploit the following observations:

- A car is typically significantly faster than the bus schedule outside of rush hour.
- Though the above is not true in rush hour, the mean distance between consecutive vehicle stops (including both traffic lights and bus stops) is significantly higher for a car than for a bus even in rush hour.
- Cars tend to not stop at bus stops.

We show empirical evidence and experiments to support the use of these features.

We use a simple confidence metric — the difference in the least squares residual fit error between the best and the second best route match — to detect ambiguity between over-

lapping bus routes and start uploading tracking data only when we are sure of the exact bus route.

We describe our algorithm in more detail below.

Input and Output. Our algorithm takes as input a sequence of location samples L_i (i = 1,2,3,...) As soon as the mode detection algorithm detects that the user is in a vehicle, the algorithm initializes and processes samples continuously while the user remains in the vehicle. The output at each time step is one of three possibilities: a bus route number and direction e.g. "49 West", a determination that the vehicle is a "Car" (i.e. cannot possibly be a bus), or "Unknown Vehicle". The algorithm keeps GPS activated as long as it is tracking a bus or is unsure; it stops sensing or uploading location information to save energy and preserve privacy as soon as it is sure the vehicle is not bus.

Algorithm Data. Our algorithm does not rely on a road map. It matches raw locations to one of several possible unique *bus shapes* defined in the general transit feed specification (GTFS) [3]. Each bus shape consists of a sequence of bus stops specified as latitude-longitude coordinates. We augment the shapes to include *relative* timing information i.e. the scheduled visit time for each bus stop on the route measured with the reference that the starting time of the trip is 0. We do not use the absolute times of each bus trip for simplicity and space reduction, though it is easy to modify the algorithm to do so. In our implementation for the city of Chicago, our augmented shape information occupies 2 Megabytes, which we believe is a reasonable footprint, and considerably smaller than the entire road map.

Initialization Period. Our algorithm begins with an "initialization period" in which we narrow down the set of candidate bus routes to match to using a geospatial index lookup. In our algorithm, the initialization phase lasts for a minute or 100 metres of covered distance, whichever comes later: empirically, we found that this is sufficient to identify the possible candidates accurately. A larger period increases the number of candidate bus route shapes by a significant factor, resulting in an unnecessary computational burden.

Subsequent location samples are processed in three steps: outlier removal, least squares minimization and post-processing to filter out non-transit vehicle traces.

Outlier Removal. Outlier removal compares each new raw data point to the preceding point, and uses a threshold on the maximum possible instantaneous speed (we use 150 km/hr). If this threshold is exceeded, the new point is discarded.

Least Squares Minimization. We use a simple online least squares minimization algorithm with an *ordering constraint* to find the closest spatial route match for a sequence of raw GPS locations.

The algorithm works as follows. Each candidate bus route consists of a sequence of *shape segments*, which are stretches of the route between consecutive bus stops. Each iteration i of the algorithm matches the input location sample L_i to one shape segment on *each* candidate route.

For each candidate route C, after having seen i location samples $(L_1, L_2, ..., L_i)$, the algorithm keeps track of two pieces of state:

- $S(C,i)$, the shape segment that location L_i was matched to on route C in the previous step.
- $ES(C,i) = \Sigma_{k=1}^{i}\ dist(L_k, S(C,k))$, the *cumulative* sum of squares of distances of location samples to their matched shape segments for each candidate route C.

At iteration $i+1$, the algorithm finds the closest matching segment $S(C,i+1)$ for location L_{i+1} that preserves the ordering constraint that this segment must occur *after* $S(C,i)$ in shape C, and computes and updates the cumulative sum-squared distance $ES(C,i+1)$.

Post-processing. After each time instant, our algorithm finds the candidate shape C_{best} with the smallest value of the cumulative sum-squared distance ES and executes a sequence of checks to determine whether the locations seen are likely to come from a bus traveling along C_{best}.

We describe these checks below in the order in which they are performed. Each check uses parameters which are knobs to the algorithm. We set these knobs using training data, as described in each section.

Spatial Distance Check. The first check ensures the sampled GPS locations align well spatially with a transit route by thresholding the root mean square error (RMSE) of fit, given by:

$$RMSE = \sqrt{\frac{ES}{N}}$$

assuming N locations have been processed so far. We use two cutoffs: $D_{maybe} = 100m$ and $D_{reject} = 200m$ in this check. We outright reject a trajectory if $RMSE \geq D_{reject}$ and stop sensing and uploading GPS locations thereafter. If the RMSE is smaller than D_{reject} but larger than D_{maybe}, we decide the route match is not a bus match for now, but continue running the algorithm. For downtown Chicago, we found the worst GPS errors to be of the order of 85 metres from our training data (Section 2.2), and hence, we set $D_{maybe} = 100$ metres, slightly higher than this number. The intuition is that a sustained mean error more than encountered in the urban canyon measurements very likely reflects a vehicle on a route that is not a bus route, and so is not a bus. We believe this parameter would be similar for urban areas of most major cities.

Schedule Deviation. Our second check captures the intuition that a vehicle traveling much faster than a bus on the best matching route is unlikely to be a bus. We compute a *deviance score* as part of our online algorithm which captures how well the time intervals between visits of the tracked vehicle to bus stops along the route line up with the predicted time intervals from the bus schedule.

We only consider visits at intervals of length at least 5 minutes to compute the deviance score. This is because over time periods smaller than 5 minutes, vehicle speeds have high variance owing to local traffic or traffic lights and tend to depart from the schedule anyway, irrespective of whether the vehicle is a car or a bus.

Suppose a location sampled at time t is matched to a shape segment S that contains a bus stop. If t is more than 5 minutes after the previous recorded visit to a bus stop (on some other segment S'), we compute the actual interval of time between these two visits. Denote this by I_{actual}. We then compare this number to the *scheduled* interval of time $I_{scheduled}$ between

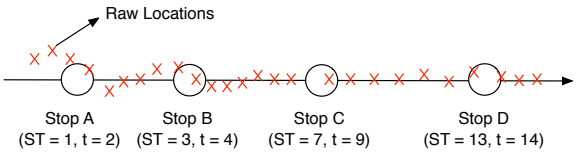

Figure 6. Computing deviance scores.

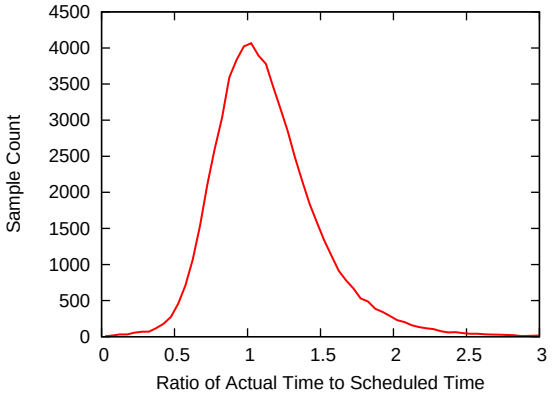

Figure 7. Histogram of Schedule Deviation Ratios, CTA Real-Time Data.

the two stops, as computed by looking up the bus schedule. We compute the ratio $r = I_{actual}/I_{scheduled}$.

Figure 6 illustrates an example. The route in the figure has four scheduled stops: A, B, C and D. The scheduled times for these stops are $ST = 1$, $ST = 3$, $ST = 7$ and $ST = 13$ respectively, all in minutes from the beginning of the route. The actual trajectory visits these stops at times $t = 2$, $t = 4$, $t = 9$ and $t = 14$ respectively. Since we only consider inter-visit intervals of at least 5 minutes, only A, C and D count as visits, and only intervals AC and CD are included in the computation. The ratio r_{AC} is $\frac{9-2}{7-1} = \frac{7}{6}$ and r_{CD} is $\frac{14-9}{13-7} = \frac{5}{6}$.

Intuitively, $r < 1$ represents the vehicle being ahead of schedule and $r > 1$ represents being behind schedule. A small value of r is likely to indicate a fast car, and a large value of r is likely to indicate heavy traffic.

Threshold on r. We found the empirical distribution of the ratio r using a training data set of 11,810 bus trajectories from CTA's real-time feed. Figure 7 shows this distribution. The graph shows that $r < 0.5$ i.e. a bus being twice as fast as the schedule, is an extremely rare event.

For the deviance check, we model the falloff on the left side in Figure 7 by a normal distribution and compute the *deviance* $N(1-r, \sigma_{fast})$ for $r < 1$. Here N is the density function for a normal distribution with mean $1-r$ and standard deviation σ_{fast}. Based on the distribution in Figure 7, we chose the constant $\sigma_{fast} = \frac{1}{6}$ to make $r < 0.5$ a 3σ event for the normal distribution, which has very low probability.

While we have limited data from cars driving along bus routes, we collected some empirical data from cars in downtown Chicago along bus routes as part of our experiments. Our data indicates that *outside of rush hour*, cars are on average twice as fast as buses. This corresponds to $r < 0.5$, and validates that our choice of σ_{fast} is likely to correctly classify most trajectories as either cars or buses.

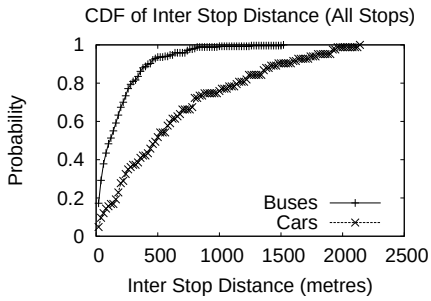

Figure 8. Inter-Stop Distance CDF for buses and cars.

For a city without a real-time feed where we would want to deploy cooperative transit tracking, we would suggest re-computing the appropriate cutoff on r using the procedure above, by collecting a training set of representative bus and car rides from that city.

Stopping Checks. The spatial error check and the schedule deviation check do not always filter out cars in rush hour, and hence we use different checks to handle this case.

We define "stops" in the raw location data as intervals of time where a vehicle remains stationary for a minimum duration (we use 15 seconds). First, we require a minimum number of stops MC to occur near known bus stops before we make a determination that the trajectory is a bus route. In our implementation, we use a cutoff of $MC = 3$ stops.

Second, we found in our experiments that the *inter-stop distance* i.e. the average distance between consecutive stops, is a good distinguishing metric between buses and cars. Figure 8 shows a CDF of the average inter-stop separation for several real bus and car traces we collected, *along the same routes*, including traces in rush hour. The figure shows that bus traces have a significantly lower mean inter-stop separation. We exploit this by adding a statistical test to our post-processing phase. We use a set of training traces to learn a mean and variance for the inter-stop distance for buses and cars respectively. At run time, our algorithm looks at the inter-stop distances actually observed on the trajectory, and determines which of the two means it more likely has.

There are two reasons for the trend in inter-stop distance — first, buses stop more often at bus stops than cars do. Second, traffic lights in many downtown areas of cities are often synchronized to allow vehicles moving at car speeds to traverse several lights without encountering a red light. Our conjecture is that buses, which service bus-stops in between lights, do not have this advantage and so tend to stop at more signals as well.

Somewhat counter-intuitively, other metrics we tried, like the actual locations of stops, or stop times, are not as good predictors as the inter-stop distance, and do not seem to be as robust to variations between rush hour and light traffic. We found the inter-stop distance to be robust across all the traces we collected, and in both rush hour and light traffic.

To understand how universal the parameters we learned are, we evaluated the inter-stop distance for two entirely different vehicular data sets. The first data set consists of over 192 hours of GPS data from shuttles operating on the University of Illinois campus, and the second data set consisted of

115 drive hours of GPS data from cars driving in the Boston area, obtained from the Cartel [13] project. The car data set had a mean inter-stop separation of 577 metres. This corroborates a mean of 634 metres found in our Chicago data set. However, the shuttle data set had a higher than expected separation of 433 metres — indicating campus shuttles operate somewhat differently from buses. Thus, the inter-stop distance parameters we learned seem accurate for cars in urban areas, but might have to be re-learned for different kinds of transit networks or for suburban areas.

Overlapping Routes. The three checks described above confirm that the tracked vehicle is likely to be a bus. The fourth and last check aims to avoid premature misclassification in the case when there are two overlapping bus routes. This is quite a common occurrence since some routes are extensions or subsets of other routes.

We compute a *confidence metric* given by the difference in the mean-squared error of fit (RMSE) between the maximum likelihood route and the second most likely bus route, computed over a 2 minute sliding window. Empirically, we have found that this confidence metric is close to zero when the algorithm is confused between two almost-identical or overlapping routes, and quickly increases when the routes diverge. When the confidence metric exceeds a cutoff, we output the bus route number, otherwise we defer judgment and report "Unknown vehicle".

A higher value for this "confidence cutoff", which we denote by *CC*, implies the algorithm is more conservative and may take more time to confirm a particular bus route and start tracking that route. A lower value enables quicker real-time tracking, but can confuse overlapping routes in some cases. We show how the choice of cutoff affects these results with experiments in our evaluation. Based on our experiments (Section 4.2), we arrived at a value of 30 metres for the confidence cutoff *CC*.

3.3 Tracking Underground Transit

As GPS receivers typically are not able to determine their location while underground, location tracking in underground environments requires non-GPS solutions. Unfortunately, WiFi and cellular localization also work poorly. In the Chicago underground transit system, we frequently observed cellular/WiFi localization errors of up to 10 miles.[2] However, in many cities cell-phone connectivity is intermittently available in tunnels, and almost universally available in stations. Thus, assuming we can estimate the location of the train through some other means, we can report the estimated location for cooperative transit tracking purposes.

We base our underground tracking system on the vehicular motion detector described in §3.1.3, and a schedule-based hidden Markov model (HMM) described below. The Viterbi algorithm is used to calculate the most likely vehicle location given past evidence.

We take the incoming stream of vehicular movement probabilities (see §3.1.3) and compute 30-second windowed means, roughly corresponding to the shortest possible stop at a station. We then apply a simple peak detector to the windowed mean probabilities. Intuitively, each interval of movement corresponds to one peak, and each interval of stationarity corresponds to one trough. We use this approach instead of absolute thresholds on the probabilities because the noise in acceleration samples varies quite a bit from setting to setting, and we found the relative trend to be a more robust estimator than the absolute value. We name these peaks and troughs *anchors*. The green line in Figure 9 shows the result of this computation for a recorded subway trace, with anchors marked. With the number of intervals determined, and with an anchor for interval, the remaining step is to find the optimal boundary between each "moving" and "stationary" interval, which can be done in a single pass of the samples between neighboring anchors. The two sets of grey boxes in Figure 9 compare the results of the algorithm (lower) to the annotated ground truth indicating when the train was in the tunnel between stations (upper). The algorithm detects all stops, and provides a good estimate of the duration of most intervals. One apparently spurious stop is detected between 600 and 800 seconds. Here, the train was actually stopped, but in the tunnel as opposed to at the station. Our subway tracking algorithm uses a schedule-based hidden Markov model to filter out this type of spurious stop, as described below.

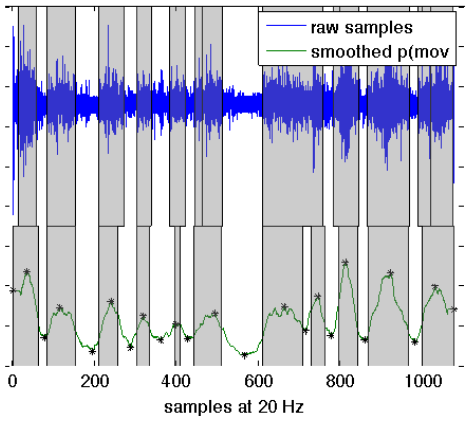

Figure 9. Detecting train mobility by accelerometer. The blue line shows raw accelerometer data, and the green line smoothed probability of being in a "moving" interval. The shaded blocks (upper) indicate actual periods of mobility, the lower indicate estimated periods.

3.3.1 Filtering out spurious stops

Given the estimated intervals of mobility and stationarity, we compute the most likely current location of the phone (train) based on the train's schedule. In contrast with the algorithm in Section 3.2, this algorithm takes the route, direction, and last GPS fix as input. Determining the route is relatively easy for subway trains, given the last GPS fix. Stations are further apart than bus stops, and many stations serve only one line. Direction of travel underground can be estimated using the compass and accelerometer together.

We briefly overview how the Viterbi algorithm works here. A Hidden Markov Model (HMM) models each observed variable — in our case, accelerometer response at

[2]On platforms that provide access to raw WiFi scans or cell ID's, it may be possible to enhance the localization technology. The iPhone platform specifically disallows this type of access.

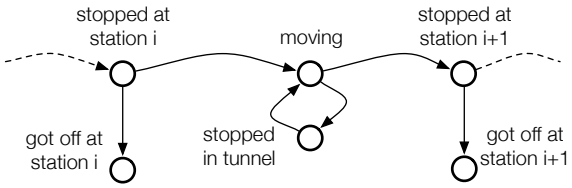

Figure 10. HMM for accelerometry and schedule-based subway tracking.

each instant of time — as coming from an underlying hidden state. A HMM has three components: a set of *hidden states*, a set of *emission scores* for each pair (P, S) where P is a point and S is a hidden state, that are proportional to the probability of observing P given underlying state S, and a set of permitted *transitions* (S_i, S_j), each with transition scores proportional to the probability of transitioning from state S_i to state S_j.

The Viterbi algorithm processes observations in an online fashion and matches them to underlying hidden states. At any time instant, it maintains a score vector SV, with one score SV_i for each hidden state i. At any time t, $SV_i(t)$ is proportional to the maximum likelihood that the current state is i given the observations up to and including time t. Given a candidate "path" i.e. a sequence of accelerations and corresponding hidden states, the likelihood of the path is the product of the emission and transition scores along the path. The maximum likelihood SV_i is the largest value of this product across all possible paths that terminate in i, and is updated at each time instant by the Viterbi algorithm. At any instant, the output is the state i with the highest score SV_i.

We now specify the states, emission probabilities and transition probabilities of our Hidden Markov Model below, a slice of which is illustrated in Figure 10.

States. The HMM has three types of states: stopped at station (one/station), moving in tunnel (one/tunnel between stations), and stopped in tunnel (one/tunnel between stations).

Emissions. The emission probabilities are straightforward: the "stopped" states always emit stationary periods, and never emit moving periods. The "moving" states always emit moving periods, and never emit stationary periods.

Transitions. All allowable transitions are represented in Figure 10. Specifically, trains cannot reverse direction (no backward edges), and cannot skip stations. Any number of stops are allowed between stations. Only two edges carry non-obvious (0 or 1) transition probabilities: those that represent transitions from a moving state to either a stopped at station state, or a stopped in tunnel state. For these, we define an arrival time probability distribution as a function of the time t spent in the moving state so far:

$$E(t_{sched} - t, \lambda_{ahead}), \qquad t < t_{sched} \tag{2}$$

$$kE(t - t_{sched}, \lambda_{behind}), \qquad t => t_{sched} \tag{3}$$

where t_{sched} the scheduled inter-station time for the pair of stations in question, and k a scaling factor to make the distribution continuous. Empirically, we set $\lambda_{behind} = 0.5$, to reflect a significant probability of late arrival, and $\lambda_{ahead} = 2$, reflecting that beating the schedule by a significant margin is unlikely.

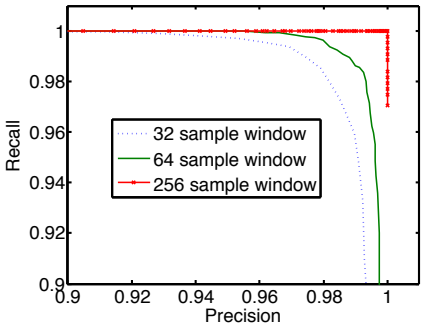

Figure 11. Precision vs. recall for 3 sample window sizes. The 256-sample detector exhibited perfect classification performance on our training set.

Transitions to "stopped in tunnel" have a low non-zero probability if $t > t_{sched}$ (the train is likely to have reached the station when it stops, so a stop is less likely to be a tunnel stop), and a higher probability of $p_{stop}(1 - E(t, \lambda_{ahead}))$ when $t < t_{sched}$, where p_{stop} is the prior probability of tunnel stops, which we set to 0.5.

4 Performance Evaluation

We base our evaluation on the following data sets:

- Accelerometer traces from 5 volunteers
- 12 hours of manually collected vehicular GPS, WiFi and cellular localization traces.
- 6,300 hours of official CTA bus tracking data.
- 192 hours of GPS data from UIC campus shuttles.
- The official Google Transit Feed Specification schedules from the Chicago Transit Authority (CTA).

The second data set was collected using applications developed in-house. In order to create a highly accurate ground-truth data set, we devised a method for entering locations manually from a map, while in motion. In a preparatory step, the operator indicates an exact location on a map. Once the location is reached, they click a 'record' button to indicate this. In the case of a stop, the button is clicked a second time to indicate the departure. Through this method, we are able to record a set of highly accurate ground truth locations. Linear interpolation is used to estimate locations between these manually recorded points.

4.1 Activity Classifier Accuracy

Using volunteers, we recorded accelerometer traces during stationary use of the phone, as well as four walking scenarios: simultaneous use of phone, holding phone in hand swinging next to the body, phone in back pocket, and phone in front pocket. Finally, we recorded accelerometry from two 30-minute bus rides, two 15-minute car drives, and a minute of waving the phone around violently, though without persistent periodicity. To create Figure 11, we computed the precision and recall values on this labeled training data, while varying the detection threshold. As the plot shows, a 256-sample window achieved perfect performance on our training set, a promising sign. By contrast, classifiers using shorter sample windows have difficulty with some samples, likely due to temporary ambiguities due to short stops,

change of position, or brief moments of suspension-induced wobble following a pothole or other road disturbance encountered while riding in a vehicle. Guided by these results, we use 256-sample windows for our remaining experiments.

	Walk	non-Walk	Walk	non-Walk
Walk	92%	8%	97.5%	2.5%
Non-Walk	0.4%	99.6%	0.1%	99.9%
	Without Peak Power		With Peak Power	

Table 3. Walking detection accuracy on a variety of labeled test traces. Peak power feature significantly improves walking detection performance. Rows represent labeled classes, columns classifier output.

The complete activity classifier consists of a decision tree, using the powers of the DFT frequency bins in the 1-3 Hz range, the variance of the sample window, and the peak frequency power as features. In contrast with [25], we do not use GPS speed as a feature, as the primary purpose of our detector is to reduce the amount of time spent using the GPS.

Table 3 compares the accuracy of our walking classifier to a non-GPS approximation of that in [25]. Results shown were produced using 10-fold cross-validation and a sliding 256-sample window, with classification performed every 32 samples (1.5 sec). We use the *classregtree* function provided by Matlab for this experiment. The 'Walk' set contains samples from 5 volunteers, walking a set course while varying the location of the phone, including in-hand in front of the body, swinging next to the body, held by the ear, in front and back pockets, as well as in backpacks and jacket pockets. The 'non-Walk' set contains several bus, train and car rides, several periods of active use without movement, biking with the phone in the front and back pockets, as well as a period of violently waving the phone around.

Classification performance is significantly improved by adding the peak power feature, resulting in 99.9% precision with 97.5% recall. We note that a determined adversary could easily fool this classifier into, for example, thinking it is walking when it is not, by simply moving the phone up and down periodically. Adversarial behavior is outside the scope of this work.

Without the peak power feature, a typical decision tree consisted of 31 nodes, with 15 internal nodes. With the feature, a typical tree had 11 nodes, with 5 internal nodes. The peak power feature was always the root node of these trees. Total spectrum power typically did not appear in any rules.

4.2 Trajectory Matching Accuracy

We have implemented the route matching algorithm described in Section 3.2 as an Android phone application. The bus schedule representation occupies approximately 2 Megabytes. Since the algorithm requires memory proportional to the number of shape segments being matched against, we use a spatial index to look up the set of relevant schedules to match to. The first few location samples are used to narrow down the set of relevant bus schedules to match to. The spatial index occupies a few extra kilobytes.

Our route matching algorithm processed approximately 6 raw locations per second on a Nexus One phone running Android 2.1, and hence can comfortably handle real-time transit matching with one sample per second.

We evaluate our transit matching algorithm on three data sets. The first set consists of 26 manually collected car and bus traces in and around downtown and suburban Chicago. The second set consists of 1,464 real bus traces spread evenly across all the bus routes in Chicago, obtained from the Chicago Transit Authority's bus data feed [2], and perturbed according to a noise model. The third set consists of 200 driving traces from shuttle vehicles that operate within the University of Illinois Campus, and have partial overlap with CTA bus routes. This set consists of 192 hours of GPS data.

We use three metrics in our evaluation: *recall*, *precision* and *decision time*. The *recall* is the fraction of bus traces for which we correctly identify the route number and start tracking the bus before the trace ends i.e. the user gets off the vehicle. Traces that are actually bus traces but we classify as a non-transit vehicle (car) or as a different bus subtract from the recall. Similarly, traces we are unsure about until the very end when the user gets off, and we report as "Unknown Vehicle" throughout, also subtract from the recall.

Given the traces that our algorithm classifies as buses and starts tracking, the *precision* is the fraction of those traces that are actually bus traces, and for which our algorithm finds the correct bus route number. Bus traces for which we find the wrong route number, or car (or shuttle) traces we think are buses, all subtract from the precision. We want our algorithm to have a high precision, as every such misclassification risks misleading users of cooperative transit tracking.

The *decision time* is the amount of time that elapses from when a rider boards a vehicle to when our algorithm is either confident that the rider is on a bus and can output a guess of the route number, or is confident that the route is not a bus and can stop tracking it. It is important to minimize the decision time when on a bus to ensure that we can use information from short bus trips (which are quite typical in downtown), and to minimize energy use and preserve privacy in the case when the user is on a non-transit vehicle.

Real CTA Data. We evaluate our algorithm on 1,464 traces from the CTA real time feed, perturbed according to a uniform noise model. We use the GPS error measurements in Section 2.2 to inform our noise model: specifically, we perturb the x and y coordinates of each location so that the mean distance from each perturbed point to the ground truth point it is derived from is 39 meters in the suburbs and 84 meters in the "loop" (downtown Chicago).

The CTA data gives us a location trace for the entire bus run. We simulate bus trips where users board a bus at a random location in the first half of the bus run. The CTA ground truth data is only sampled every minute, and we interpolate intermediate points. As a consequence, we cannot use the minimum stop count criterion *MC* and the inter-stop distance criteria mentioned earlier — we disable these checks. This is not a problem for route accuracy evaluation; for evaluating how well we can distinguish buses from cars (following subsections), we re-enable these checks, since we are testing on fine-grained one-sample-per-second GPS data that includes stop information.

Figure 12 shows the precision and recall for the CTA data

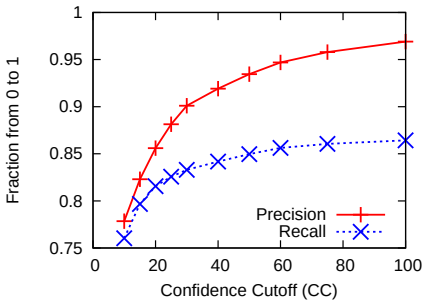

Figure 12. Prec./recall vs Confidence Cutoff, CTA Data.

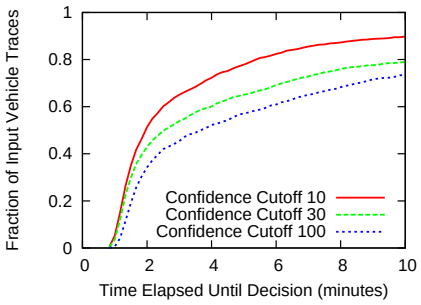

Figure 13. CDF of Decision Time, CTA Data.

set, for different values of the confidence cutoff *CC* discussed in Section 3.2. Figure 13 shows a CDF of the decision time for different values of *CC*.

We see that the precision increases significantly with *CC*. This is because most of the classification error in the CTA bus data set comes from mistaking one route for an overlapping, but slightly different route. Increasing *CC* requires the algorithm to have more confidence before it confirms an answer, and reduces the error rate.

However, the recall, which is the fraction of buses identified correctly, does not improve as dramatically as the precision. This is because although higher *CC* improves precision, it also causes the algorithm to fail to report an answer for many bus trips before the user gets off since it does not have sufficient confidence.

The decision time (Figure 13) is smallest for a low value of *CC* (e.g. $CC = 10$) since the algorithm requires less confidence to come to a decision. However, this value has poor precision. For a high value like $CC = 100$ metres, the precision is very good (97%) but the decision time can be quite high in the tail. From an overall system point of view, taking both the recall and decision time results into account, we found $CC = 30$ to be the best option that maximizes overall recall for a given elapsed time.

For our chosen value of *CC*, the median decision time is less than 3 minutes, and we can detect 55% of buses correctly within 5 minutes (The 55% number comes from multiplying the value of the decision time CDF at 5 minutes with the recall). We only classify 9% of bus trips wrongly, and most of these are cases with two overlapping routes, one being a proper subset of the other. Many users will not be affected significantly by these errors. For example, CTA buses 108 and 8A have virtually identical routes, stopping at almost all of the same stops. Reporting a 108 bus as an 8A would not affect most users.

In addition to the above traces from the CTA, which were sampled once a minute and interpolated, we also manually collected 16 bus traces sampled once a second. The algorithm classified 13 of them correctly and missed classifying 3 of them. Two of these were relatively short traces with ambiguity between two candidate bus routes, such that even one of our authors could not tell the difference. The "confidence cutoff' check for overlapping routes failed in both cases.

Postprocessing Evaluation. Our evaluation above showed that on real bus data, our algorithm can quickly come to a decision with good accuracy. However, for a complete evaluation of the system, we also need to evaluate how well the algorithm performs on car and non-transit vehicle traces — does it quickly reject these traces and stop tracking the user?

Route	Verdict	Reason For Verdict
Along 12W	Car	Too fast, schedule dev. $< -3\sigma_{fast}$
Along 156N	Car	Too fast, schedule dev. too high
Along 156S	Car	Too fast, schedule dev. too high
Along 8N	Car	Interstop dist. of 530m too large
Along 8S	Car	Interstop dist. of 628m too large
Near 22N	Unkn.	Only 1 stop near a bus stop

Table 4. Transit matching on car traces along or near known bus routes.

We cannot directly evaluate how well our algorithm filters out car traces since we only have access to a limited number of car traces. Rather, we evaluate this indirectly using two smaller-scale experiments, as explained below.

In the first experiment, we ran our algorithm on 200 GPS traces (amounting to over 192 drive hours of data) from shuttles operating on the University of Illinois Campus near downtown Chicago, that are *not* part of the CTA bus network. This is a challenging data set because many of the shuttle routes have significant overlap with some CTA bus routes. On this data set, our algorithm with $CC = 30$ performed well, classifying 194 of the 200 traces correctly as non-transit vehicles. 6 vehicle traces were classified incorrectly as CTA buses, owing to significant overlap with CTA routes. The minimum bus stop count check ($MC \geq 3$) that we require before we report a vehicle as a bus turns out to be crucial here. The shuttle stops do not coincide with CTA stops. Without the bus stop check, as many as 90 of the 200 traces were classified incorrectly as CTA routes. The median time to a decision for the shuttle traces was 4 minutes. Thus, on average, we would stop tracking the user after 4 minutes and save energy/preserve privacy.

In our second experiment, we collected a limited data set of car traces in downtown Chicago, some carefully chosen to coincide with bus traces. Table 4 shows the result of running our transit matching algorithm on these traces. Some of these traces were deliberately chosen to coincide exactly with bus routes while others were in the vicinity of bus routes.

The table shows that our postprocessing checks work correctly to distinguish these cars from buses, even though 5 of the 6 traces coincide exactly with bus routes and the route matching algorithm determines them to have low spatial er-

ror. In three cases, the trace is classified as a car because the vehicle is too fast i.e too far ahead of the bus schedule to be a bus. The 8N and 8S cases are interesting because they were both collected during rush hour, and the traces are not significantly faster than the bus schedule. However, their inter-stop separation is of the order of 500-600 meters, much higher than the mean inter-stop separation for buses which is approximately 200 meters. This causes them to be correctly classified as cars.

As control experiments, we also collected bus traces along routes 8, 22 and 156 in both directions at approx. the same time as the car traces. All of these passed all the post-processing tests and were correctly classified as buses.

Finally, in addition to the traces shown in the table, we collected 4 car traces in the vicinity of major bus routes, but not coinciding with them. All 4 were classified as cars because the spatial error is large. Specifically, these traces had root mean square distances of 798, 241, 863 and 129 meters from their closest matching bus routes, all larger than our cutoff error $D_{maybe} = 100$ meters.

4.3 Tracking Accuracy in Subway

The purpose of the subway tracking algorithm is to determine the arrival time of a train at each station. To the ability of the subway tracking algorithm of determining the current location of the train, we collected a new, annotated set of four new traces from Chicago CTA Red and Blue line subways. At each station, we recorded the arrival time using an iPhone app designed for this purpose, while continuously recording accelerometer data. The phone was in active use during this recording, with no special precautions taken. We first evaluate the performance of the mobility detector on two of these traces, and then the localization performance of the algorithm as a whole.

Figure 14 shows the mobility detection results for two subway traces. The blue line shows the raw accelerometer magnitude, and the green line the window-averaged probability of mobility. The gray boxes on top show the actual time spent in tunnels between trains, and the lower gray boxes show the detected intervals of vehicular movement. While detected mobility matches up well with the ground truth in general, the occasional spurious stop does appear.

To evaluate the performance of the subway tracking HMM and viterbi algorithm, we compare the estimated arrival time at each station with the actual arrival time. We also compare with the raw train schedule (offset by our starting time) as a straw-man. Under ideal conditions, the train would follow the schedule perfectly, with no errors as a result. On a normal run, therefore, we would expect the schedule to be highly accurate; we noticed nothing out of the ordinary during these runs. Figure 15 plots, for each of the four runs (BS, BN, RN, RS), the estimated, scheduled and actual arrival time at each station, with station numbers offset for readability. We note that while there is significant variance, our estimator tends to stay closer to the actual arrival time than the schedule does, even on these uneventful runs. Across all runs and stops, the mean difference between the estimated and actual time was 41 seconds, 18 seconds median. By contrast, the difference between the scheduled and actual time was 55 seconds mean, 38 seconds median.

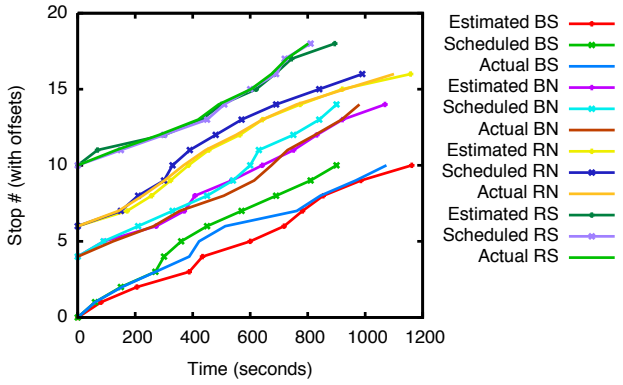

Figure 15. Comparison of estimated, scheduled and actual arrival time at each station. Station numbers offset for readability.

4.4 Utility of Cooperative Transit Tracking

To evaluate the overall utility of cooperative transit tracking, we developed a simulator based on the CTA General Transit Feed Specification (GTFS) data. This dataset contains the locations of bus and train stops, and the shapes of bus and train routes. By overlaying the stop information onto the known route shape data, we implemented a functional transit network model that simulates a bus traveling along a particular shape. To realistically simulate the movement of buses in our simulator, we collected 6,300 drive hours of real-time location information from the buses in the CTA network. We then combined the real-time data with known stop and shape information to interpolate high-frequency "real-time" bus location points along each route.

We compared the performance of a simple statistical arrival-time prediction algorithm against the CTA's published arrival-time schedule. The prediction algorithm is a table-based predictor based on the mean inter-stop travel time for all pairs of stops on each route, computed separately for every hour of the week, to account for typical fluctuations in traffic levels, such as rush hour traffic. [14] evaluated the performance of different bus-arrival prediction schemes, and found that the historical travel time model performed well.

We obtained recent ridership statistics from the CTA that specify the distribution of bus riders across the CTA routes, which we use in our simulation.

In order to evaluate cooperative transit tracking, we randomly simulate hypothetical users that would query the system for an arrival time. We assign each hypothetical user a bus route at random based on the distribution from the CTA ridership statistics. We then randomly select a bus stop, direction and time at which the user would want to travel on that route.

We ask the cooperative transit tracking system for the current (if presently tracked) or last-known (if not presently tracked) location of the bus, and then feed its location into our table-based predictor to determine the next time at which a bus with the specified route number will arrive at the selected stop. We assume the user will arrive at the stop at the time predicted by our system and wait for the vehicle to arrive. Thus, the difference between this prediction and the actual arrival-time as supplied by the real-time CTA data, is

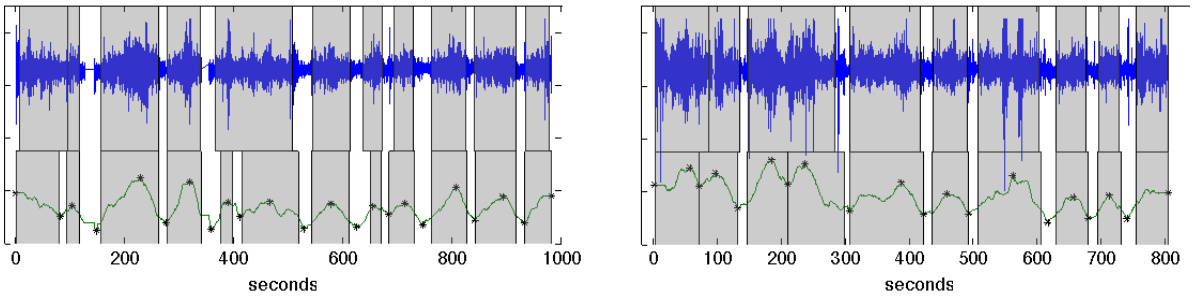

Figure 14. Visual representation of the results for the CTA blue line going south, and red line going north. Upper grey boxes are ground-truth time in tunnel, lower grey boxes are estimated times of mobility.

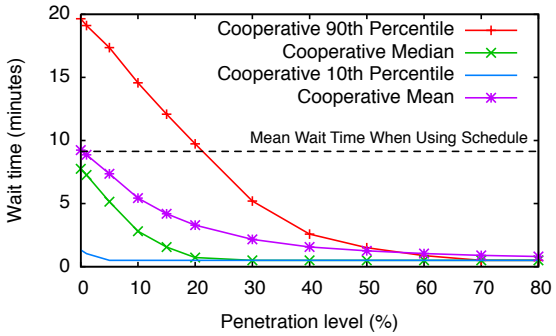

Figure 16. Wait time vs. Penetration level using Cooperative Transit Tracking with fallback on schedule.

computed as the expected waiting time for the user.

4.4.1 Cooperative Transit Tracking Simulation

Our system simulates the bus that will be servicing the selected stop based on our real-time trace data. We begin the simulation with the bus at its origin location, and proceed to follow it through time to its final stop. We assume users instrumented with our smartphone application board and alight the vehicle based on an exponential distribution.

Our exponential distribution is seeded by a mean time between boardings (i.e., inter-arrival time) for instrumented transit riders. The mean of this distribution is calculated from the route ridership reported by the CTA. We assume CTA riders are equally distributed across all trips the selected bus route takes, and we assume a constant percentage of the riders have our smartphone app installed.

We take into account both the route matching accuracy and the decision time needed by our system to correctly identify the vehicle, before tracking begins. We do this by using the empirical distribution of decision times from our results (§4.2). For each simulated rider instrumented with our app, we use data from the rider with probability equal to the system recall. We then sample a decision time from the distribution in Figure 13. When a hypothetical user queries out system, we check to see if an instrumented rider is on-board the vehicle, and has been on-board for a time exceeding this decision time. If yes, we return the current location of the vehicle. Otherwise, we return its last known user-tracked location.

4.4.2 Results

Using our simulation environment, we evaluate the performance of cooperative transit tracking as a function of the "penetration level", which is the percentage of all transit riders who are instrumented with our smartphone application. The main results from our simulation efforts are shown in Fig. 16. As the penetration level rises, the amount of time spent waiting for the bus at the bus stop, or "wait time" is reduced. The reported wait time is the number of minutes spent at the bus stop, if you arrived at the stop when either the schedule or the tracking system said the bus would arrive. [3] Specifically, for a bus that is real-time tracked all the way to the stop, we report a "wait time" of 30 seconds, a reasonable safety margin. The motivation behind this interpretation of wait time is that users are likely to adapt their travel behavior when they have real time information, and plan to arrive at the stop with little or no margin. Similar gains could be achieved without real-time tracking, but only if buses always arrived on time.

In 16, the system falls back on the schedule when no real time tracking is available. Thus, at the 0% penetration level, performance is identical to simply using the schedule (and arriving two minutes early). Conversely, at 100% penetration, performance is essentially identical to an officially installed transit tracking system. We note the median wait time drops sharply even at low levels of penetration, and that at 20% penetration level, the median wait time is essentially zero.

In Figure 17, we show the wait times for only those users that were able to take advantage of cooperative transit tracking data. This illustrates how the quality of our arrival time predictions improve as the penetration level rises. Here, every request is served using cooperative transit tracking data, but the quality of the arrival time prediction depends on when the location of the bus was last reported.

Finally, Fig. 18 shows the proportion of requests served using cooperative transit tracking data as a function of the penetration level. The proportion rises steeply up to the 20% penetration level, where more than 83% of requests are served. Intuitively, above the 20% level the vast majority of vehicles either already have an instrumented rider aboard, or

[3] When using the schedule only, we found arriving two minutes early to be ideal: sometimes buses arrive ahead of the schedule, and getting to the stop exactly on the scheduled time would often incur a long wait for the next bus.

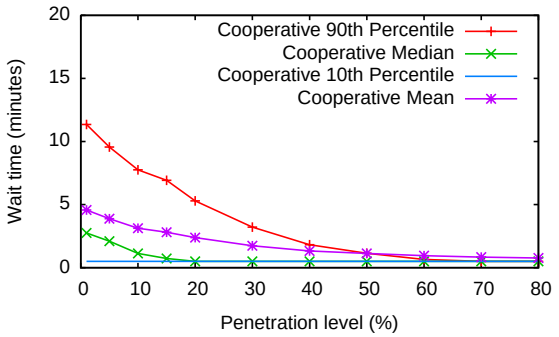

Figure 17. Wait time vs. Penetration level using Cooperative Transit Tracking Only.

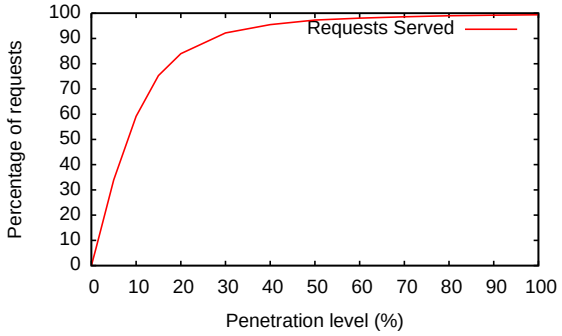

Figure 18. Requests served vs. Penetration level.

had one aboard quite recently.

These results are particularly encouraging in the context of trains, where a relatively small number of vehicles serve a very large number of users. Chicago's CTA has real-time tracking for all its buses, but does not supply it for its trains, which serve nearly 600,000 rides per day.

5 Conclusion

We have described a system for cooperative transit tracking that combines power-efficient activity detection using accelerometer data, memory-efficient spatio-temporal bus trajectory matching using least squares minimization, and accelerometry in conjunction with a Hidden Markov model to track underground trains when other localization schemes do not work. Our end-to-end results suggest that with a penetration level of 5%, our system can have a significant impact on commuter wait times. We find that cooperative transit tracking is a viable alternative where official solutions are too expensive or otherwise difficult to put in place.

6 References

[1] CTA Bus Tracker to Expand System-wide. `http://www.chicagobus.org/news/bus-tracker-expands-systemwide`.

[2] CTA Real-Time Feed. `http://www.ctabustracker.com`.

[3] General Transit Feed Specification (GTFS). `http://code.google.com/transit/spec/transit_feed_specification.html`.

[4] NextBus To Roll-out System Wide In San Francisco. `http://www.nextbus.com/corporate/press/index.htm#muniNew`.

[5] Nike + iPod. `http://www.apple.com/ipod/nike/run.html`.

[6] M. S. Braasch. Multipath effects. In B. W. Parkinson and J. J. Spilker, editors, *Global Positioning System: Theory and Applications*, pages 547–566. American Institute of Aeronautics & Astronautics, 1996.

[7] J. Bussmann, W. Martens, J. Tulen, F. Schasfoort, H. van den Berg-Emons, and H. Stam. Measuring daily behavior using ambulatory accelerometry: The activity monitor. *Behavior Research Methods, Instruments & Computers*, 33(3):349–356, 2001.

[8] T. Choudhury, G. Borriello, S. Consolvo, D. Hähnel, B. Harrison, B. Hemingway, J. Hightower, P. V. Klasnja, K. Koscher, A. LaMarca, J. A. Landay, L. LeGrand, J. Lester, A. Rahimi, A. Rea, and D. Wyatt. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2):32–41, 2008.

[9] I. Constandache, R. R. Choudhury, and I. Rhee. Toward mobile phone localization without war-driving. In *INFOCOM*. IEEE, 2010.

[10] T. Denning, A. Andrew, R. Chaudhri, C. Hartung, J. Lester, G. Borriello, and G. Duncan. Balance: Towards a usable pervasive wellness application with accurate activity inference. In *HotMobile*, 2009.

[11] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *IPSN '05*, page 70. IEEE Press, 2005.

[12] A. El-Rabbany. *Introduction to GPS: the Global Positioning System*. Boston, MA: Artech House, second edition, 2006.

[13] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. ACM SenSys*, Nov. 2006.

[14] R. Jeong. *The prediction of bus arrival time using automatic vehicle location systems data*. PhD thesis, Texas A&M University, 2004.

[15] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *MobiSys*, 2008.

[16] M. B. Kjaergaard, J. Langdal, T. Godsk, and T. Toftkjaer. Entracked: Energy-efficient robust position tracking for mobile devices. In *MobiSys*, 2009.

[17] A. Le Faucheur, P. Abraham, V. Jaquinandi, P. Bouye, J. L. Saumet, and B. Noury-Desvaux. Study of human outdoor walking with a low-cost gps and simple spreadsheet analysis. *Medicine and Science in Sports and Exercise*, 39(9), 2007.

[18] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Pervasive*, pages 1–16, 2006.

[19] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *IJCAI'05*, pages 766–772, 2005.

[20] J. Lester, P. Hurvitz, R. Chaudhri, and G. Borriello. Mobilesense - sensing modes of transportation in studies of the built environment. In *UrbanSense*, 2008.

[21] L. Liao, D. Fox, and H. Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *International Journal of Robotics Research*, 26(1):119–134, 2007.

[22] K. Lorincz, B. rong Chen, G. W. Challen, A. R. Chowdhury, S. Patel, P. Bonato, and M. Welsh. Mercury: a wearable sensor network platform for high-fidelity motion analysis. In *SenSys*, 2009.

[23] B. Nham, K. Siangliulue, and S. Yeung. Predicting mode of transport from iphone accelerometer data. Tech. report, Stanford Univ., 2008.

[24] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman. Activity recognition from accelerometer data. *American Assoc. for AI*, 2005.

[25] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using mobile phones to determine transportation modes. *Transactions on Sensor Networks*, 6(2), 2010.

[26] E. I. Shih, A. H. Shoeb, and J. V. Guttag. Sensor selection for energy-efficient ambulatory medical monitoring. In *MobiSys*, 2009.

[27] A. Thiagarajan, L. Sivalingam, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan. Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones. In *SenSys*, 2009.

[28] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys*, 2009.

[29] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W.-Y. Ma. Understanding transportation modes based on gps data for web applications. *ACM Trans. Web*, 4(1):1–36, 2010.