VTrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones

Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts,
Samuel Madden, Hari Balakrishnan
MIT CSAIL
{arvindt,lenin,katrina,madden,hari}@csail.mit.edu

Sivan Toledo
Tel-Aviv University
stoledo@tau.ac.il

Jakob Eriksson
U. Illinois Chicago
jakob@cs.uic.edu

Abstract

Traffic delays and congestion are a major source of inefficiency, wasted fuel, and commuter frustration. Measuring and localizing these delays, and routing users around them, is an important step towards reducing the time people spend stuck in traffic. As others have noted, the proliferation of commodity smartphones that can provide location estimates using a variety of sensors—GPS, WiFi, and/or cellular triangulation—opens up the attractive possibility of using position samples from drivers' phones to monitor traffic delays at a fine spatiotemporal granularity. This paper presents VTrack, a system for travel time estimation using this sensor data that addresses two key challenges: energy consumption and sensor unreliability. While GPS provides highly accurate location estimates, it has several limitations: some phones don't have GPS at all, the GPS sensor doesn't work in "urban canyons" (tall buildings and tunnels) or when the phone is inside a pocket, and the GPS on many phones is power-hungry and drains the battery quickly. In these cases, VTrack can use alternative, less energy-hungry but noisier sensors like WiFi to estimate both a user's trajectory and travel time along the route. VTrack uses a hidden Markov model (HMM)-based map matching scheme and travel time estimation method that interpolates sparse data to identify the most probable road segments driven by the user and to attribute travel times to those segments. We present experimental results from real drive data and WiFi access point sightings gathered from a deployment on several cars. We show that VTrack can tolerate significant noise and outages in these location estimates, and still successfully identify delay-prone segments, and provide accurate enough delays for delay-aware routing algorithms. We also study the best sampling strategies for WiFi and GPS sensors for different energy cost regimes.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Algorithms, Measurement

Keywords

GPS, Location, Localization, Traffic, Congestion, Mobile, Phone, Car, Transportation, Energy, Sensor

1 Introduction

Traffic congestion is a serious problem facing the road transportation infrastructure in many parts of the world. With close to a billion vehicles on the road today, and a doubling projected over the next decade [22], the excessive delays caused by congestion show no signs of abating. Already, according to much-cited data from the US Bureau of Transportation Statistics, 4.2 billion hours in 2007 were spent by drivers stuck in traffic on the nation's highways alone [3], and this number has increased by between 3$\times$ and 5$\times$ in various cities over the past two decades.

As has been observed previously, *real-time traffic information*, either in the form of travel times or vehicle flow densities, can be used to alleviate congestion in a variety of ways: for example, by informing drivers of roads or intersections with large travel times ("hotspots"); by using travel time estimates in traffic-aware routing algorithms to find better paths with smaller expected time or smaller variance; by combining historic and real-time information to predict travel times in specific areas at particular times of day; by observing times on segments to improve operations (e.g., traffic light cycle control), plan infrastructure improvements, assess congestion pricing and tolling schemes, and so on. An important step for all these tasks is the ability to *estimate travel times* on segments or stretches of the road network.

Over the past few years, the idea of using *vehicles as probes* to collect traffic data has become popular [1, 17, 20]. Here, vehicles equipped with GPS-equipped embedded computers log the current time and position periodically as they travel, sending this data to a server over a wireless network. This approach is better than flow-monitoring sensors (e.g., inductive loops) deployed on the roadside because vehicles can cover large areas more efficiently.

While the dominant source of such data today is from commercial fleets which already have GPS devices for fleet management, the proliferation of *mobile smartphones* opens up the attractive option of obtaining a massive amount of data directly from end users [1, 21, 4, 20]. Unlike professional fleets (and trucks, which contribute to much of the currently available commercial data), end users travel on the roads and at the times that are most useful to monitor for the purpose of reducing end-user commute duration. Smartphones are equipped with multiple position sensors including GPS, WiFi, and cellular radios; these can be sampled to obtain timestamped position estimates that can be delivered to a server, and processed to estimate driving times on different road segments at a fine spatio-temporal granularity.

However, using smartphones as traffic probes poses two significant challenges:

1. *Energy consumption:* Sampling accurate positions from GPS at a fine granularity (e.g., every second) consumes significant amounts of energy on some phones. Sampling GPS less frequently, or using a noisy sensor like WiFi can consume much less energy (Section 5.4). For a desired level of accuracy, it is important to choose the sensor(s) to sample and the sampling rate appropriately to minimize energy consumption.[1]
2. *Inaccurate position samples:* Even when energy consumption isn't a concern, GPS isn't always available. Some phones have WiFi and cellular radios but no GPS. GPS experiences outages on some phones when in a user's pocket, and in "urban canyons" near tall buildings or tunnels. These, and energy concerns, force the use of lower energy sensors such as WiFi or cellular radios, which are accurate only to tens or hundreds of meters — the closest road in the map to a position sample is often not the road that a vehicle actually drove on, making travel time estimation challenging.

As with many sensor applications, using smartphones as traffic probes raises some privacy concerns, which are out of scope for this paper; see Section 6 for a discussion of other work that addresses this concern.

We present VTrack, a real-time traffic monitoring system that overcomes these challenges. VTrack processes streams of timestamped, inaccurate position samples at time-varying periodicity from mobile phones. It uses a Hidden Markov Model (HMM) to model a vehicle trajectory over a block-level map of the area. VTrack performs *map matching*, which associates each position sample with the most likely point on the road map, and produces travel time estimates for each traversed road segment. VTrack provides real-time estimates recent to within several seconds to users. It also compiles a database of historic travel delays on road segments.

A key contribution of our paper is an extensive evaluation of VTrack on a large dataset of GPS and WiFi location samples from nearly 800 hours of actual commuter drives, gathered from a sensor deployment on 25 cars. The data was gathered from two sources: an iPhone 3G application, and from embedded in-car computers equipped with GPS and WiFi radios.

[1]If all users keep their phones charged while driving, then energy isn't a concern, but we believe that imposing that constraint is an unreasonable barrier to deployment.

The main question we investigate is how the quality of VTrack's travel time estimates depends on the sensor(s) being sampled and the sampling frequency. We built two applications using VTrack to measure the quality of time estimates. The first application reports *hotspots* (roads with travel times far in excess of that expected from the speed limit), with a view to avoiding them. The second is a route planner which finds shortest expected time paths using the segment travel times estimated by VTrack. Our key findings are:

1. *HMM-based map matching is robust to noise,* producing trajectories with median error less than 10% when sampling only WiFi, as well as for up to 40 meters of simulated Gaussian noise.
2. *Travel times from WiFi localization alone are accurate enough for route planning, even though individual segment estimates are poor.* When location samples are very noisy, it is difficult to attribute a car's travel time to small stretches of road—for example, time estimates from WiFi for individual segments have a median error of 25%. However, somewhat counter-intuitively, using these times to find shortest paths works well—over 90% of shortest paths found using WiFi estimates have travel times within 15% of the true shortest path. This is because groups of segments are typically traversed together, and our estimation scheme ensures that errors on adjacent or nearby segments "cancel out." *Moreover, estimating real drive times actually matters—for congested scenarios, using just speed limits to plan paths yields up to 35% worse than optimal paths.*
3. *Travel times estimated from WiFi localization alone cannot detect hotspots accurately, due to the outages present in WiFi data.* We find that our hotspot detection algorithm misses many hotspots simply because of a lack of data on those segments. This problem is not as apparent in route planning, since in that scenario we are focused on choosing a path that has a travel time closest to the shortest path, rather than worrying about particular segments. However, on the subset of segments for which we *do* have WiFi data, we are able to accurately detect more than 80% of hotspots, and flag fewer than 5% incorrectly.
4. *When GPS is available and free of outliers, sampling GPS periodically to save energy is a viable strategy for both applications.* On our data, for up to $k = 30$ seconds (corresponding to roughly 2 or 3 road segments), sampling GPS every k seconds produces high quality shortest paths, assuming GPS is always available. If the phone is also WiFi-equipped, the tradeoff between sampling GPS every k seconds, sampling WiFi or a hybrid strategy depends on the energy costs of each sensor (discussed in Section 5).

Using HMMs for map matching is not a new idea [16, 19]. However, to the best of our knowledge, previous work has focused on map matching frequently-sampled GPS data with low noise, and on qualitative studies of accuracy. Our key contribution is a quantitative evaluation of the end-to-end quality of time estimates from noisy and sparsely sampled locations, both important in the energy-constrained smartphone setting.

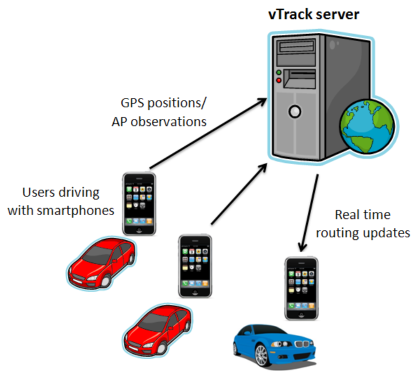


Figure 1: Architecture

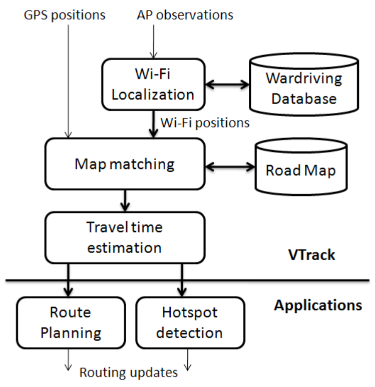


Figure 2: VTrack Server

2 Overview and Challenges

Figure 1 shows the architecture of VTrack. Users with mobile smartphones run an application that reports position data to the server periodically while driving. VTrack currently uses GPS and WiFi position sensors. The server runs a travel-time estimation algorithm that uses these noisy position samples to identify the road segments on which a user is driving and estimate the travel time on these segments. The estimates are then used to identify hotspots and provide real-time route planning updates.

Figure 2 shows the VTrack server in detail. In cases where GPS is not present, not working, or too power-hungry, VTrack uses WiFi for position estimation; access point observations from WiFi in smartphones are converted into position estimates using a localization algorithm, which uses a "wardriving" database of GPS coordinates indicating where WiFi APs have been observed from in previous drives. Positions from these sensors are fed in real-time to our estimation algorithm, which consists of two components: a *map-matcher*, which determines which roads are being driven on, and a *travel-time estimator*, which determines travel times for road segments from the map-matched trajectory.

2.1 Applications

We want to support two key applications using real-time travel times:

Detecting and visualizing hotspots: We define a "hotspot" to be a road segment on which the observed travel time exceeds the time that would be predicted by the speed limit by some threshold. Hotspots are not outliers in traffic data; they occur every day during rush hour, for example, when drivers are stuck in traffic. Our goal is to detect and display all the hotspots within a given geographic area which the user is viewing on a browser. This application can be used directly by users, who can see hotspots on their route and alter it to avoid them, or can be used by route avoidance algorithms, to avoid segments that are frequently congested at a particular time.

For hotspot detection, we want travel time estimates to be accurate enough to keep two metrics low: the *miss rate*, defined to be the fraction of hotspot segments that we fail to report, and the *false positive rate*, the fraction of segments we report as hotspots that actually aren't.

Real-time route planning: With the exception of hotspots, users are generally more concerned about their total travel time, rather than the time they spend on a particular road. Route planning that minimizes the expected time is one way for users to find a good route to a particular destination. Real-time route planning is especially useful because it allows users to be re-routed mid-drive.

Our goal is to provide users with routes that minimize travel times. Ultimately, the routing scheme will have to take travel time *prediction* into account. In this paper we only focus on estimation, a necessary step for any prediction scheme. To analyze our travel time estimates in the context of route planning, we study how much worse the shortest paths we find are compared to the true shortest paths (shortest in terms of travel time).

Our goal is to run these applications both on websites and on users' phones. We envision a user with an app running on his or her phone showing position, nearby traffic, and offering a navigation service, while continuously sending position estimates back to our servers. We have built a prototype of such an application for the iPhone. We also have a website where users can view current traffic delays and receive route planning updates. Figure 3 shows a screenshot of our website (the iPhone app is very similar).

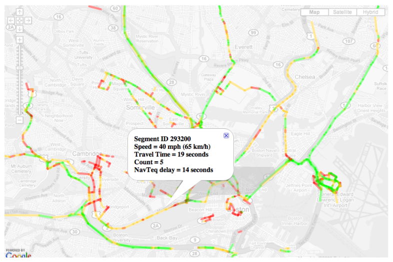

Segment ID 293200
Speed = 40 mph (65 km/h)
Travel Time = 19 seconds
Count = 5
NavTeq delay = 14 seconds

Figure 3: Web application showing traffic delays.

2.2 Requirements and Challenges

In the context of the above applications, we need map-matching and time-estimation algorithms that are:

1. *Accurate*, such that time estimates are close enough to reality to be usable for route planning and hotspot

detection. For route planning, we believe that errors in the 10-15% range are acceptable; at most a 3 to 5 minute error on a 30 minute drive. Some current route planners use road speed limits scaled by some constant factor to estimate times; our evaluation (Section 5) shows that these estimates can be significantly inaccurate in congested scenarios.

2. *Efficient enough to run in real time* as data arrives. Some existing map-matching algorithms run A*-style shortest path algorithms multiple times per point, which we found to be prohibitively expensive and hard to optimize.
3. *Energy efficient*, such that the algorithms use as little energy as possible while meeting accuracy goals, to maximize lifetime on battery-powered phones. Sampling GPS every second may perform well, but is of no use if it excessively drains the phone's battery.

Meeting these constraints presents several challenges:

Map matching with outages and errors. Though algorithms for matching GPS traces to road segments exist [16, 19], previous work focuses on frequent samples of low-error-data, and hasn't measured how well these algorithms work on WiFi, nor how well they work on large numbers of traces collected from real urban environments with tunnels and tall buildings. Characterizing the accuracy of these algorithms in a real-world setting is essential to determine if applications like route planning and hotspot detection are feasible.

Time estimation—even with accurate trajectories—is difficult. When source location data is very noisy, even if the map-matching algorithm is able to determine the correct route, it is difficult to attribute a car's travel time to a particular road segment on that route.

Localization accuracy is at odds with energy consumption. GPS is accurate to within about 5 meters in many settings, but is power hungry: our measurements, and those of others [9] indicate that sampling GPS consumes far more energy than sampling WiFi (up to 20x more). WiFi is less accurate, typically producing location estimates within a mean radius of about 40 meters of the true location, and only providing coverage in locations where there are APs (Figure 4). Also, GPS isn't perfect; our experiments with the iPhone show it does not work when in a driver's pocket while driving (even if a satellite fix has been acquired first). It also doesn't work in tunnels, and experiences outages and large errors in urban canyons formed by tall buildings in cities.

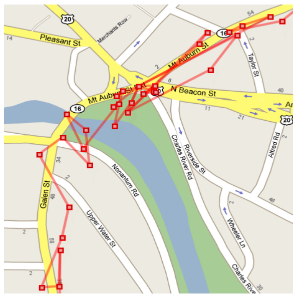


Figure 4: A WiFi trace; verticles represent locations estimated from historical observations of WiFi routers.

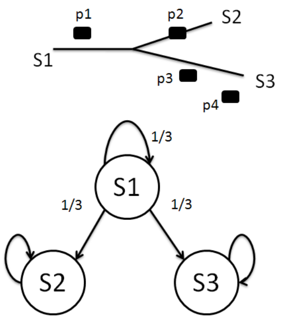


Figure 5: Hidden Markov Model Example

3 VTrack Algorithms

Map matching is the process of matching a sequence of noisy or sparse position samples to a sequence of road segments. It is essential for travel time estimation, as users are interested in the time it takes to traverse a particular road segment or sequence of road segments.

If a phone is using its GPS sensor, it can report its position estimate directly. However, if it is using its WiFi sensor to localize, it reports the access points (APs) which it observes, and these need to be converted into position estimates. We do this by using a wardriving database which maps APs to positions. In VTrack, we have deployed a system of 25 cars equipped with a wireless card and a GPS device, producing a database that maps from APs to the locations where they were observed. A single AP can be mapped to multiple GPS points; we compute the location of that AP as the centroid of these points. Then, given a collection of APs that a car saw at a given time, we estimate the car's position as the centroid of the locations of those APs.

An obvious choice for map matching is to map each position sample to the nearest road segment. However, we show in Section 5.5 that this scheme fails even in the presence of small amounts of noise. As a result, we—and many others [16, 19]—use a Viterbi decoding over a *Hidden Markov Model (HMM)* (described below) to estimate the route driven. However we differ from previous approaches in two key ways: how we handle outages and how we model transitions between road segments. We also pre-process the position samples to remove outliers and post-process the HMM output to remove low quality matches. This prevents us from producing inaccurate travel time estimates.

We first briefly explain Hidden Markov Models and how they apply in the context of map matching.

3.1 HMM and Viterbi Decoding

An HMM is a Markov process with a set of hidden states and observables. Every state emits an observable with a particular conditional probability distribution called the emission probability distribution. Given some input, an HMM traverses its states to produce its output: the observables emitted at each state. While the output (list of observables) is known, the sequence of states is not; the problem is to determine the most likely sequence of states that produced the output.

Transitions among the hidden states are governed by a different set of probabilities called transition probabilities. In the

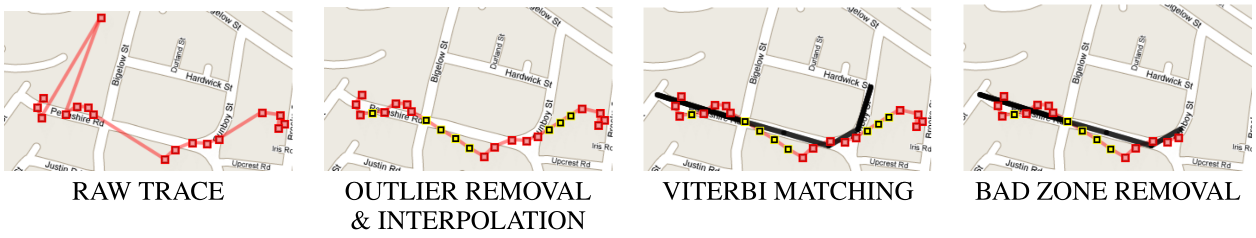


Figure 6: The Map Matching process. A raw location trace is gradually refined to produce a final, high quality street route. In this example, due to noise and outages, only the first three segments produced quality estimates.

map matching context, the hidden states are road segments and the observables are position samples. Here, the emission probability for a given ⟨segment, position⟩ pair represents the probability of seeing that position conditioned on the vehicle being on that segment. The transition probability is the probability of transitioning (driving) from one segment to the next. Our challenge, then, is: given a sequence of positions, determine the most likely sequence of road segments—the *route*—that corresponds to these positions.

Viterbi decoding [24] is a dynamic programming technique to find the maximum likelihood sequence of hidden states given a set of observables and the emission probability distribution and transition probabilities. In our case, the hidden states corresponds to the route, and the observables are the position samples.

Figure 5 shows an example where S1, S2, and S3 are road segments and p1, p2, p3, and p4 are position samples. There is an equal probability of transitioning from S1 to S1, S2, or S3. Because the emission probability density function is a decreasing function of distance, assuming the transition constraints as shown in the state diagram, the maximum likelihood sequence of segments for the given sequence of position samples is S1, S3, S3, and S3. Although p2 is closer to S2, the most probable hidden state of the point is S3 given the transition constraints.

It is clear from this example why an HMM makes sense for map matching: it is robust to position samples that lie closer to one road segment than the one from which they were observed, and is able to capture the idea of a continuous route instead of a sequence of segments.

3.2 Map Matching

Figure 6 illustrates our map matching approach. Prior to using an HMM, we pre-process the data to eliminate outliers and cope with outages. We say a sample p is an *outlier* if it violates a speed constraint; that is, for some threshold speed $S_{outlier}$, the car would have had to travel faster than $S_{outlier}$ mph to travel from the previous sample to p in the observed time between p and the previous sample. We chose $S_{outlier} = 200$ mph, about twice the maximum expected speed on a freeway. This threshold is intentionally conservative and accommodates for the speed between subsequent noisy GPS or WiFi samples.

After outlier removal, we use a simple but novel scheme to deal with outages. Previous work either does not consider outages [16], or deals with them by doing multiple computationally expensive shortest path calculations, or by invoking a route planner [19] to find the transitions in the HMM. Our algorithm uses a simple and efficient scheme which inserts *interpolated* points in regions where the location data experiences an outage. The algorithm generates interpolated samples at 1 second intervals along the line segment connecting the last observed point before the outage and the first following the outage, assuming a constant speed along this line. These interpolated points are then fed to the HMM along with the sampled points. Empirically, we have found that the HMM matches a straight-line interpolation to an approximate shortest path quite well, and achieves good map-matching accuracy (see Section 5.5). The outlier removal and interpolation steps are shown in the second panel of Figure 6; the interpolated points are shown in green.

Once outliers have been removed and interpolation applied, the Viterbi algorithm is used to predict the most likely sequence of road segments corresponding to the observed and interpolated points (third panel of Figure 6). The hidden states in the Markov model that we use are directed road segments, and observations are position samples. The algorithm computes the most likely sequence of road segments given the observations and outputs a road segment for each position sample. These road segments specify the route that was taken.

Finally, bad zone removal (fourth panel of Figure 6) is applied to remove low-confidence Viterbi matches (e.g., from very noisy data). This is described in more detail later.

We now explain the transition probabilities which govern the possible sequences of roads, and the emission probability, which governs the most likely road segments for a particular point to be observed from.

Transition Probabilities. Our transition probabilities reflect the following three notions: 1) For a given road segment, there is a probability that at the next point, the car will still be on that road segment. 2) A car can only travel from the end of one road segment to the start of the next if it uses the same intersection (we also take into account one-way streets); this ensures that the output road segments are continuous. 3) A car cannot travel unreasonably fast on any segment.

The transition probability p from a segment i at sample $t-1$ to a segment j at sample t is as follows:

1. If $i = j$, $p = \varepsilon$ (defined below).
2. If j does not start where i ends, $p = 0$.
3. If j does start where i ends, $p = \varepsilon$ or 0 (this reflects the third notion, and is explained below).

We keep all of the non-zero transition probabilities constant in order to avoid preference for routes with low-degree segments (routes with intersections without too many outgoing roads). The alternative, which is used by several other map matching schemes [16], is to partition one unit of probability between all the segments that start at the end of i. This results in higher transition probabilities at low-degree intersections than at high-degree intersections, which is not a faithful model of the underlying process. In fact, we found that in the presence of noise, such an approach performs poorly (Section 5.5). To avoid this, we ensure the non-zero transition probabilities are constant across *all segments*, but at the same time sum to 1 for a given segment. To do this, we use a dummy "dead-end" state $\emptyset$ and a dummy observable $\perp$. We set ε to be $\leq 1/(d_{\max}+1)$, where $d_{\max}$ is the maximum number of segments that start at the end of the same segment (the maximum out-degree of the graph). We set the transition probability from i at $t-1$ to $\emptyset$ at t so that the sum of the transition probabilities for segment i at t comes out to 1. The transition probability from $\emptyset$ at $t-1$ to $\emptyset$ at t is 1 (thus effectively assigning zero probability to all paths that transition to $\emptyset$).

The third item in our transition probability definition reflects the fact that we want to prohibit cars from traveling unreasonably fast on any segment. If we are considering a transition from segment i to segment j, our algorithm calculates the time it would have taken the car to travel from i to j (based on the times at which the positions were observed). If this time implies that the car would have had to travel at higher than a threshold speed ($S_{outlier}$), we say that this transition is impossible and assign it a probability of 0; otherwise, the transition is possible and we assign it a probability of ε. As mentioned before, we use a relatively relaxed value of 200 mph for $S_{outlier}$ to avoid over-constraining the HMM in the presence of noisy position estimates.

Emission Probabilities. Our emission probabilities reflect the notion that it is likely that a particular point was observed from a nearby road segment, but not necessarily the closest one. Concretely, the emission probability density of segment i at position sample ℓ is $N(\text{dist}(i,\ell))$ where N is a Gaussian function with zero mean, and $\text{dist}(i,\ell)$ is the Euclidean distance between i and ℓ. The variance of N depends on the sensor that produced the sample; we use different variances for GPS and for WiFi because they have different error distributions. For GPS we used a variance that translated to 10m of noise (as measured in previous studies of GPS accuracy), and for WiFi a variance that translated to 50m of noise (this number was obtained from our empirical data).

Viterbi algorithm. We use Viterbi decoding [24] to determine the most likely hidden states in the HMM; this corresponds to the most likely sequence of road segments traversed by the user. The third panel in Figure 6 shows the most likely route in black. Note that the output is actually a sequence of $\langle$point, road segment$\rangle$ pairs, where consecutive points can lie on the same road segment.

Bad Zone Detection. After running Viterbi, we remove zones of bad map matching before performing travel time estimation. If a position sample is matched to a segment which is at a distance greater than a threshold from the sample, we tag it as a bad match. We use a conservative threshold of 100 meters to accommodate for approximately twice the maximum expected noise (from WiFi). We also tag the subsequent points in the forward and the backward direction until the distance between the position samples and their matches begins to increase again. This removes sequences of points which cause peaks in the distance function, and intuitively captures the notion that when a position sample is incorrectly mapped, it causes its surrounding samples to also be incorrectly mapped, and hence the entire zone needs to be removed.

4 Travel Time Estimation

The output of map matching is the most likely segment that each point in the raw trajectory came from. Hence, the traversal time $T(S)$ for any segment S consists of three parts:

$$T(S) = T_{left}(S) + T_{matched}(S) + T_{right}(S)$$

$T_{left}(S)$ is the time between the (unobserved) entry point for S and the first observed point (in chronological order) matched to S. $T_{matched}(S)$ is the time between the first and last points matched to S. $T_{right}(S)$ is the time between the last point matched to S and the (unobserved) exit point from S.

As stated in Section 3, map matching adds interpolated points to ensure that points in the output are high-frequency (typically separated by one second, and guaranteed to be separated by at most two seconds). Hence, if map matching outputs a continuous sequence of segments, both $T_{left}(S)$ and $T_{right}(S)$ are upper-bounded by 1 second, and for segments that are not too small, $T_{matched}(S)$ is the main determinant of delay. This makes time estimation easy: we first assign $T_{matched}(S)$ to the segment S. We then compute the time interval between the first point matched to S and the last point matched to S_{prev}, the segment preceding S in the map match, and divide it equally[2] between $T_{right}(S_{prev})$ and $T_{left}(S)$, and similarly for the segment S_{next} following S.

Map matching does not always produce a continuous sequence of segments because bad zone detection removes low confidence matches from the output. We omit time estimates for segments in, immediately before and after a bad zone to avoid producing estimates known to be of low quality.

4.1 Estimation Errors

The main source of error in our time estimates is inaccuracy in the map matched output, which can occur for two reasons:

Outages during transition times. Transition times are times when the car is moving from one road segment to another. Without observed samples at these times, it is impossible to determine the delay on each segment exactly. While map matching can use interpolation to determine the correct sequence of segments, accurate delay estimation for individual segments is harder than just finding the correct trajectory. For example, we cannot know whether the car waited for a red light at the end of one segment, or crossed the intersection quickly but slowed down at the beginning of the next segment.

Noisy position samples. Suppose that the location of a car is sampled just after it enters a short length segment, but a noisy sensor (like WiFi) estimates the location of the car to be

[2]How we divide does not affect estimates significantly because the interval is bounded by 1 second.

near the end of the segment. If this is the only sample matched to that segment, the location error is likely to translate into an extremely inaccurate delay estimate. In particular, we found that determining travel times for *small segments*, with lengths of the order of the magnitude of noise in the location data, is nearly impossible. For example, if we only get a location sample every 60 seconds, in the absence of a sophisticated model for car speed, it is usually impossible to determine the driving time on a group of whole segments to within less than 60 seconds, even if location samples are completely accurate. Similarly, if a 100 meter location error is fairly likely, it is hard to estimate accurate delays for segments whose size is comparable to 100 meters.

Although we are unable to estimate accurate times or speeds for individual small segments, we show in Section 5.2 that estimation errors on adjacent or nearby segments tend to have *opposite* signs because the overall trajectory is typically correct. Because groups of segments (representing commonly driven sub-routes) tend to be reused repeatedly in trajectories, our *end-to-end* travel time estimates for routes as well as groups of segments are much more accurate, and adequate for route planning and hotspot detection (Section 5).

5 Evaluation

We evaluate VTrack on a large data set of GPS and WiFi location estimates from real drives, obtained from Cartel [15]. We first describe our data set, and the procedure we used to clean and obtain reasonable ground truth for the drives. We then evaluate the quality of travel time estimates for shortest path routing and hotspot detection, for different combinations of GPS and WiFi sampling rates. We show that we can achieve good accuracy for these applications using WiFi localization or sparsely sampled GPS, saving energy in both cases. We drill down into the results and show that this accuracy is achieved *in spite of large travel time estimation errors on individual segments.* For cases when both GPS and WiFi are available, we discuss the accuracy-energy tradeoff between using WiFi, using sparsely sampled GPS, and hybrid schemes that sample both, for the energy cost scenarios estimated in Section 5.4. Finally, we assess the impact of noise other than that from WiFi data using simulations, and show that our HMM-based map matching algorithm is robust to significant amounts of simulated Gaussian noise.

5.1 Data and Method

5.1.1 Ground Truth

Obtaining ground truth is a fundamental challenge in the evaluation of any travel time estimation system. Approaches such as recording additional data in test drives ([25] makes video recordings to record road segment transitions, e.g.) are accurate but very costly, leading to relatively small amounts of test data. Another approach is to use GPS samples as ground truth [5]. This works well most of the time, but fails in regions where GPS is subject to errors and outages (e.g., near large buildings and in tunnels), making it inappropriate to use such data for ground truth.

As others have noted previously [19], it is impossible to get perfect ground truth. We use a different approach based on aggressive data cleaning to produce ground truth with reasonable confidence for a subset of our drives. The steps to clean the data are as follows:

1. For each GPS point g in a drive, we consider the set of segments S_g within a 15 meter radius of g (we picked 15m because it is the width of a typical road segment).
2. We search the space of these segments to match the sequence of points g to a continuous sequence of segments X_g, such that each $X_g \in S_g$. Therefore, each GPS point is matched to one of its neighbors found in the previous step. We throw out all points g which cannot be matched in this way (for example, if there are no segments within a 15 meter radius).
3. We now look for outages of more than a certain duration (we used 10 seconds, approximately the median traversal time for a segment) and split the drive into multiple drives on either side of the outage.
4. Finally, we project each g to the closest point on X_g to obtain a corresponding ground truth point g'.

The output traces from the data cleaning process, which we term *clean drives*, satisfy three constraints: no gap exceeds 10 seconds, each GPS point is matched to a segment at most 15 meters from it, and the resulting segments form an unbroken drive. We believe that these constraints taken together define a subset of the GPS data that can be treated as ground truth with high confidence[3].

Validation. We attempted to validate our ground truth for delays on a very small data sample (about 1 hour of driving) by means of a simple field drive in the Boston area. We used an Android phone equipped with GPS to continuously record the location of the phone, and a phone application to mark the locations of turns. A human operator pressed a button whenever the vehicle crossed an intersection, signal or stop sign. We compared the travel times between successive turns (i.e., button presses) obtained from the application (which is as close to ground truth as human error would permit) to that obtained from the cleaned version of the GPS data, cleaned as described above. The average error in travel times from the cleaned version was 4.74% for one such half hour drive in Boston with approximately 30 segments, and 8.1% for another half hour drive in Cambridge with approximately 50 segments – and manual inspection reveals that a significant portion of this error is likely human error in marking exactly where a turn is.

We use the clean GPS drives with associated WiFi samples to evaluate the accuracy of time estimates for each of the following sensor sampling strategies:

1. **Continuous WiFi.** We simply map match the WiFi data, compute travel times from the match, and compare to times from the clean drives.
2. **GPS every k seconds.** We sample the GPS data at intervals of k seconds, discarding $k-1$ samples and outputting every k^{th} sample. We do not directly feed every k^{th} GPS sample because our cleaning procedure is already biased towards drives where GPS is close to the drive. To overcome this, we add Gaussian noise with a variance of approximately 7 meters to every k^{th} GPS

[3]It is certainly possible that systematic errors in GPS could cause this approach to fail, but we assume these are rare.

sample (7m Gaussian noise is widely acknowledged to be a reasonable model for GPS [8]).

3. **GPS every k seconds + WiFi in between.** This is a combination of the sampling strategies above.
4. **Scaled Speed Limits.** This approach does not use sensors. It simply uses speed limits from the NAVTEQ road database [2] to produce static time estimates for road segments. Since drivers do not drive exactly at the speed limit, directly using speed limits suffers systematic bias. To overcome this, we scaled all the speed limits by a constant factor k, and chose k to peg the mean difference between time estimates from speed limits and time estimates from our ground truth to zero. We found the value of k to be close to 0.67, reflecting that drivers in our data drove at 67% of the speed limit on average.

The advantage of our approach is that it is cheap because it only uses easy to collect GPS data, but realistic, because it restricts our evaluation to drives with near-perfect travel time data. Importantly, we don't test the system on these nearly-perfect samples (which would introduce bias by making GPS look artificially good), but instead test on noisy versions of the samples to simulate real GPS performance. A limitation is that we do not model outliers in GPS, because we do not have any form of ground truth for regions where GPS fails. Hence all the results in our paper involving GPS are for the *outlier-free* case which assumes GPS is always available and only has a small amount of Gaussian noise. When GPS has significant outliers (e.g., urban canyons or tunnels), using WiFi may be preferable.

5.1.2 Raw Data

We began with a collection of 3998 drives collected from 25 cars equipped with GPS and WiFi sensors in an urban area. The traces contain simultaneous GPS and WiFi location estimates. The GPS data consists of raw readings. The WiFi location estimates were produced using centroids of access point observations, as described in Section 3. We cleaned the raw data using the procedure described previously; in addition, we also omitted traces shorter than 2 km, with fewer than 200 samples (about 3 minutes) or fewer than 10 road segments, and portions of traces where a car traveled slower than 2 km/h for 60 or more consecutive seconds, which we interpreted as parked (typical traffic signals do not last more than this, but this number could be modified to accommodate longer signal wait times). The result was a data set of 2145 drives, amounting to nearly 800 hours of driving. Figure 7 shows the coverage map of our evaluation data, i.e., the set of distinct road segments on which our vehicles drove.

5.2 Route Planning

We evaluate the time estimates produced by VTrack for different combinations of GPS and WiFi sampling rates, in the context of route planning. In route planning, we are interested in finding the best routes between source-destination pairs in the road network that optimize some metric. We choose a popular metric: minimizing the expected drive time.

For each sensor setting, our evaluation takes as input a set of clean drives D_{gt} and a corresponding set of noisy or subsampled drives D_{noisy} (for example, using WiFi, or infrequently-sampled GPS, or a combination). We run the

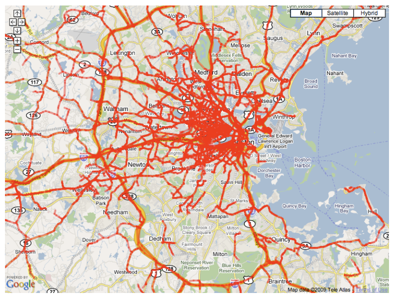


Figure 7: Coverage Map Of Our Evaluation Drives.

VTrack algorithm on D_{noisy} to produce travel time estimates for each segment on those drives. Our goal is to understand how errors in these estimates from VTrack affect the quality of the shortest path estimate, i.e., how much worse the paths the route planner finds using these inaccurate travel times are, compared to the best paths it can find from ground truth travel times.

To measure this, we consider the set of road segments for which we have a ground truth travel time estimate from at least one clean drive in D_{gt} (call this set S_{gt}). We construct the *induced graph* G_{gt} of the entire road network on the set S_{gt} (i.e., the subset of the road network defined by the segments with ground truth and their end points). Note that G_{gt} in general may be disconnected: if this is the case, we divide it into connected components. We pick a connected component at random and simulate a drive between a random source-destination pair within that component[4]. For each segment S in the graph, we pick two weights:

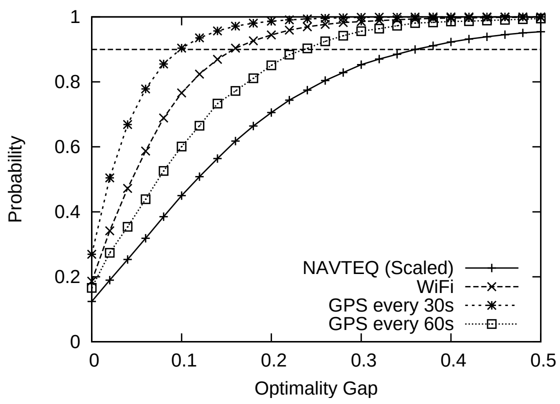
1. A ground truth weight for S picked from some clean drive D_{gt} in which S appears.
2. An estimated weight, picked to be the VTrack travel time estimate for S from the noisy or sparse version of D_{gt}, D_{noisy}. This is available whenever VTrack includes S in its estimated trajectory for D_{noisy}. If the VTrack trajectory for D_{noisy} omitted S, we fall back on estimating the travel time using the scaled speed limit for the segment obtained.

We now run Dijkstra's algorithm on G_{gt} with the two different weight sets to find two paths, P_{gt} and P_{noisy}. To evaluate the quality of our route planning, we compare the *ground truth times* for these two paths (which are always available, because we use the induced graph) and compute the following "optimality gap" metric for each source-destination pair:

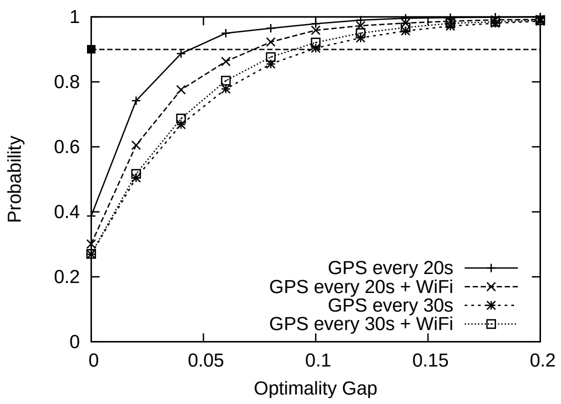
$$\mathit{Optimality\ Gap} = \frac{\mathit{Time}(P_{noisy}) - \mathit{Time}(P_{gt})}{\mathit{Time}(P_{gt})}$$

Figure 8(a) shows CDFs of the optimality gap across 10,000 randomly selected source-destination pairs, for different combinations of sensors and sampling rates, as well as for a strawman which performs route planning using just scaled

[4]We divided the graph into zones and picked the pair from different zones to ensure drives with a minimum length of 2km.



(a) GPS vs WiFi vs NAVTEQ.



(b) GPS vs Hybrid GPS + WiFi.

Figure 8: CDF of Optimality Gap when route planning using VTrack. Larger optimality gaps are worse.

speed limits (this is similar to what some route planning systems do today). We believe that an optimality gap of up to 10-15% is reasonable, corresponding to a path no worse than 35 minutes when the shortest path is 30 minutes. Figure 8(b) shows the same CDF, but comparing the strategy of sampling GPS every *k* seconds and interpolating in between, to using WiFi estimates in between.

First, we see that travel times from WiFi localization alone are good enough for route planning. The 90th percentile optimality gap is 10-15% for WiFi, implying that 90% of simulated commutes found paths that were no worse than 10-15% compared to the optimal path.

Similarly, travel times from GPS sampled every 30 seconds or slightly more are good enough for route planning. The 30 second number seems to reflect that interpolating GPS works very accurately for up to 3 or 4 missing road segments.

Both the above strategies significantly outperform using scaled speed limits. Using speed limits works reasonably well in the median, but incurs a significant tail of poor path predictions (all in congested scenarios, as one would expect).

Third, a hybrid strategy using both subsampled GPS every 30 seconds, and WiFi in between improves performance over just subsampling GPS or just using WiFi, but the gains are small compared to the energy costs, as we see later. Note that using just GPS every 20 seconds with interpolation in between outperforms sampling WiFi in between GPS samples (but not for 30 seconds), suggesting that interpolating GPS works better than WiFi over a time scale of 20 seconds (one or two road segments).

For sampling frequencies beyond 60 seconds, route planning starts to perform poorly, with at least 10% of commutes finding paths more than 20-25% worse than optimal. This suggests that sampling GPS less often than a minute (approximately half a kilometer or more, at city driving speeds) is unlikely to yield good route predictions, at least for urban areas.

We now drill down and show that, somewhat counter-intuitively, WiFi localization works adequately for route planning *in spite of large estimation errors on individual road segments* (this is also true of GPS subsampling). Figure 9 shows a CDF of estimation errors (relative to ground truth travel time) on individual road segments when using only WiFi localization. WiFi localization has close to 25% median error and 50% mean error. However, the errors in the CDF are actually two-sided because the Viterbi algorithm when run on WiFi finds the *correct trajectories* quite often, and only mis-assigns points on either side of segment boundaries. Hence, errors on groups of segments traversed together on drives tend to cancel out, making end-to-end estimates superior to individual segment estimates. For example, we might distribute a travel time of 30 seconds as 10 seconds to segment *A* and 20 seconds to an adjacent segment *B* when the truth is 20:10, but if *A* and *B* are always traversed together, this error doesn't affect the end-to-end estimate.

Figure 10 confirms this intuition. This graph shows the CDFs for two map-matching metrics on just WiFi data: *Point Error Rate (PER)*, the fraction of position samples assigned to the wrong segment, and *Segment Error Rate (SER)*, a (normalized) edit distance between the map-matched output and the ground truth trajectory. We see that map matching performs better on *SER* than on *PER*, confirming that the trajectories are often correct, and the matching of points to segments within a trajectory is more often wrong.

The optimality gap metric we have presented captures the impact of incorrect predictions, as well as missed segments in prediction (because we fall back on a less accurate NAVTEQ delay when we miss a segment in map matching), but not that of *spurious segments*, which are segments produced by the map matching but not in the ground truth. We omit details here, but we found that the number of spurious segments is less than 15% of the number of ground truth segments for most approaches, including WiFi alone, and for sampling GPS every 20 or 30 seconds.

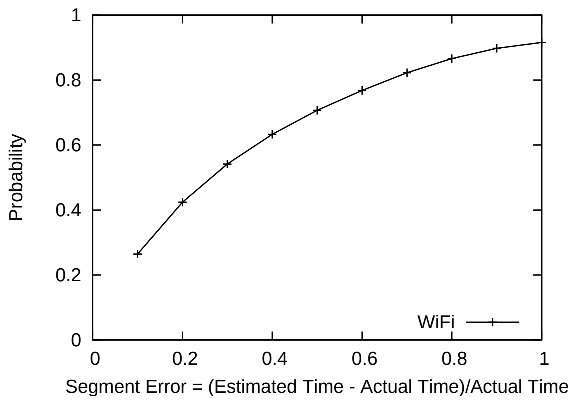


Figure 9: CDF of PER and SER for WiFi localization.

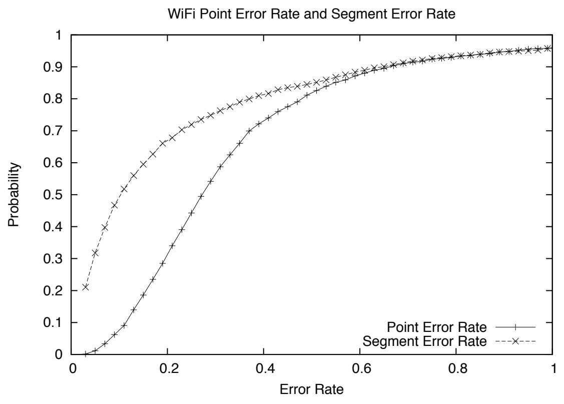


Figure 10: Map matching errors for WiFi localization.

5.3 Hotspot Detection

We also evaluate VTrack's time estimates in the context of hotspot detection, i.e., detecting which road segments are highly congested so that drivers can avoid them. We say a road segment has "high delay" if the observed travel time on that segment differs from the travel time estimated with scaled speed limits by at least *threshold* seconds. The hotspots for a particular threshold value are the set of road segments which have high delay based on the ground truth data, and we are interested in examining how many of those hotspots we can detect using traces of WiFi data, GPS+WiFi data, or subsampled GPS data.

Note that an alternative definition of a hotspot is a road segment in which the observed travel time is more than *threshold times* the travel time estimated with scaled speed limits. We found that this definition was unfair due to its dependence on the length of the segment: small segments, which have a low estimated travel time, would be more likely to be flagged as hotspots than large segments, which have a high estimated travel time. Thus, we chose to use the first metric, as it more accurately reflects the road segments that drivers view as hotspots.

To detect hotspots using a trace of WiFi, GPS+WiFi, or subsampled GPS data, we first find the segments that have high delay. We classify each of these segments as a hotspot, as well as their two adjacent segments if these segments appear in the trace. Effectively, this results in us flagging "groups" of segments as hotspots. This method reflects the fact that our travel time estimation algorithm is not always certain as to which segment a high delay should be attributed to, but tends to only be off by one segment (in a sequence).

To measure our performance, we use two metrics: success rate and false positive rate. The success rate represents how many hotspots we successfully found, and is defined as the fraction of ground truth hotspots that our algorithm detected. The false positive rate is the fraction of hotspots we "detected" that were not actually hotspots. In the real-world, this is equivalent to suggesting that a driver avoid a segment that is not actually congested. We record a false positive if we flagged a group of segments as a hotspot, but that *group* was not a hotspot (we can define a group as a hotspot analogously to a segment: the group is a hotspot if the total travel time on the group is more than *threshold* $\times$ *number segments in group* seconds above the travel time estimated by scaled speed limits).

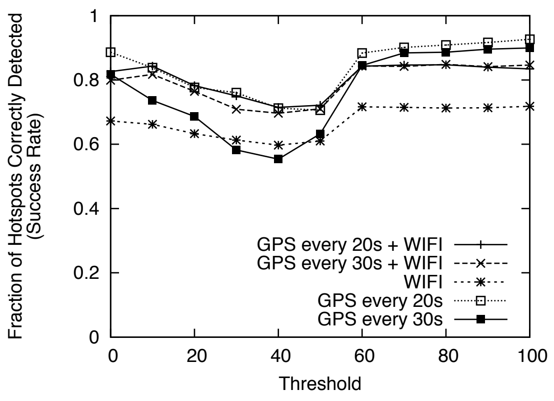


Figure 11: Success Rate of Hotspot Detection

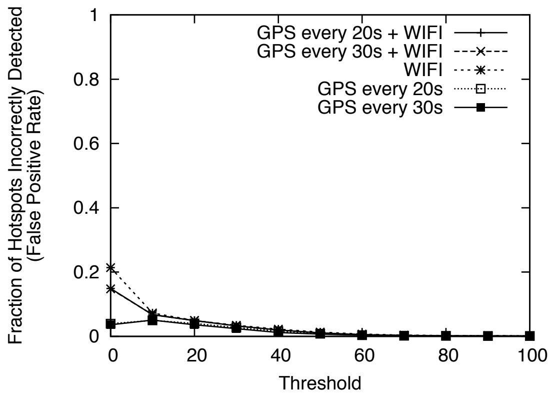


Figure 12: False Positive Rate of Hotspot Detection

Figure 11 shows the success rates for each strategy, and Figure 12 shows the false positive rates for each strategy.

There are a few interesting points to make from these graphs.

First, for the strategies involving GPS, the success rate is consistently above .8, and many times around .9. This means that these strategies can consistently detect between 80% and 90% of hotspots. The success rate for WiFi data is much worse, frequently around .65. This reflects the fact that there are significant WiFi outages, and our hotspot detection algorithm cannot find a hotspot for which it has no data. In contrast, our GPS data has complete coverage. We do note that if we restrict our statistics to road segments where there *is* WiFi data, WiFi has a success rate comparable to all of the GPS schemes.

In all schemes, the false positive rate remains low. This indicates that our hotspot detection algorithm is not too aggressive, in that it does not flag segments that are not hotspots as such. This is a desirable property as we do not want users to avoid road segments that are not actually congested.

It is interesting to note that, with the exception of GPS every 30 seconds, our success rate in every strategy remains relatively consistent across all threshold values, indicating that our algorithm is robust to applications which have specific requirements for what constitutes a hotspot. Our false positive rate also remains low for most threshold values, and for the most part only increases for small thresholds. This is due to the fact that, with a small threshold, we are likely to flag many groups of segments as hotspots, and these groups may include some segments that do not have high delay (but were included because they were adjacent to a segment with high delay).

We note that there is a dip in all of the strategies at around 40 seconds. At small thresholds, we flag many segments as hotspots, and thus have a high success rate (as well as more false positives). As the threshold begins to increase, we start to miss some hotspots, but our false positive rate decreases dramatically. This explains the portion of the graph before a threshold of 40.

The second portion of the graph can be explained by examining the total number of hotspots. As the threshold increases, the number of hotspots naturally decreases. At about 40 seconds, the rate of decrease slows, and from 60 seconds on, the number of hotspots remains fairly constant. This means that many of the road segments that are hotspots with a threshold of 60 are also hotspots with a threshold of 100; their observed time differs from their estimated time by over 100s. As a result, we do very well flagging hotspots at larger thresholds, since they are "obvious" hotspots, in some sense.

Discussion of WiFi Outages. We found that WiFi data has an outage rate of 42% (i.e., 42% of the time which we are trying to use WiFi data, we do not get a WiFi observation). This raises the question: how can a sensor that is so unreliable still perform well in some applications? In particular, although WiFi sensors did not work particularly well for hotspot detection, they did work well for route planning. The reason for this is that in route planning, using the scaled speed limit estimates on segments where there is no WiFi data is generally sufficient. Outages in WiFi tend to cause missed data points on small segments; these are exactly the segments where NAVTEQ estimates are reasonable. Even though using NAVTEQ estimates on the entire path does not perform well (they cannot account for congestion), using these estimates as a back-up for segments missing WiFi data can work well in certain cases. In hotspot detection, on the other hand, we can never use the scaled speed limit estimates in place of WiFi data. After all, we define a hotspot as a road segment where the observed time estimate *differs* from the scaled speed limit estimates.

5.4 Energy vs Accuracy

In this section, we combine the results presented above with an empirically determined model of energy costs for WiFi and GPS, to study the tradeoff between energy and accuracy for three different strategies: GPS subsampled periodically, WiFi, and a hybrid strategy that combines the two. As before, all our results involving GPS assume GPS is available at all times and free of outliers.

5.4.1 Energy Measurements

To get a better understanding of energy costs of location-estimation on a smartphone, we measured the power consumption of GPS and WiFi on the iPhone using battery life as an indicator. We wrote a simple iPhone application that repeatedly requests location estimates at either GPS or WiFi accuracy, with a user-specifiable periodicity, until the battery drains to a fixed level from a full charge (in our experiments we drain the battery to 20%). Because of the way iPhone applications work currently, we had to run this application in the foreground with the phone's screen turned on, so we ran a third control experiment and measured the total lifetime with the screen on, but without requesting location estimates. Our results are summarized in the following table:

Location Mechanism	Sampling Period	Lifetime
None	-	7 h
GPS	continuous (1/sec)	2 h 24 m
GPS	30 sec	2 h 27 m
GPS	2 min	2 h 44 m
WiFi	continuous (1/sec)	6 h 30 m

These results show that on the iPhone, GPS is extremely power hungry and reducing the sampling period does not improve the performance greatly. The reason appears to be that the iPhone OS leaves the GPS on for about a minute even when an application de-registers for position estimates; hence, sampling GPS every 30 seconds is as expensive as continuous sampling, and sampling once every two minutes doesn't save a significant amount of power. In contrast, the iPhone seems to do a much better job of aggressively managing WiFi power consumption when no data is being sent and the radio is being used only for localization. Previous work [9] corroborates these numbers on a different device, showing that WiFi localization can provide 2-3 times the battery life of GPS localization on Nokia N95 phones.

Since poor GPS power management appears to be an iPhone artifact, it is instructive to estimate sensor energy costs on a hypothetical device with better GPS power management and the ability to run a localization program in the background (with the screen turned off). There is every reason to believe that future platforms will provide both of these features.

Estimating WiFi Cost. We can use the numbers in the table to solve for the WiFi energy cost as a fraction of the continuous GPS sampling cost. Suppose the battery has capacity c Joules. The baseline lifetime, without any location estimates, is 7 hours (25,200 seconds). Therefore, the baseline (screen) power consumption $b = c/25200$ Watts. The lifetime with both GPS and the screen on is 8,640 seconds, so $g + b = c/8640$. Solving, we get $g = c/13147$. On doing a similar analysis for WiFi, we get $w = c/327360$ and $g/w = 24.9$—that is, the cost per sample of GPS is 24.9× the cost per sample of WiFi. This also suggests that continuous WiFi sampling is about 8 percent of the total power consumption when the phone is running continuously with the screen on (since $w/g = .08$), which means that a background application that continuously samples WiFi is unlikely to dramatically alter the battery life of the phone.

Estimating GPS Subsampling Cost. The GPS implementation on the iPhone is not optimized for power consumption, so sampling it infrequently does not actually save energy. The iPhone does not duty cycle the GPS as aggressively as it should, perhaps to avoid the latency (and potential additional power cost) of acquiring a GPS fix when powered back on. In theory, most GPS chipsets claim that they can acquire a "hot fix"—a location estimate from a powered-down state when the GPS has not moved very far or been off for very long and has a reasonable estimate of the position of satellites—in about 1 second. To measure how well current GPSes actually perform in a setting where have direct control over the GPS power, we used a standalone Bluetooth GPS unit with an MTK GPS two-chip chipset (MTK MT3301 and MT3179). This device has a power switch. We measured the time for the device to power on and acquire a fix after it had previously obtained a "hot fix" location estimate. We found that with a power-off time of anywhere from 5 seconds to 30 minutes, it took the GPS receiver about 6 seconds to power on, acquire a satellite fix, and report the first reading over Bluetooth to a laptop computer. Hence, a sampling strategy that acquires a GPS fix every $k > 6$ seconds using hot fixes should use approximately $\frac{6}{k}$ of the power of continuous sampling.

5.4.2 Discussion

On the iPhone, sampling the GPS at frequencies up to two minutes per sample uses a large amount of energy. Based on Figure 8(a), it is clear that, for the iPhone, if battery lifetime is a concern, WiFi sampling is the best strategy. Sampling GPS slower than every two minutes doesn't make sense as it results in a higher optimality gap than only WiFi, and using GPS faster than that drains the battery quicker.

These numbers also show that, for our prototype iPhone application using WiFi estimation, we would use about 1/6th of the total charge of the phone if it were run by a user for an hour a day with the screen on (since $c/25200W + c/327360W \times 3600s = .15c$) , which seems to be a reasonable level of consumption for most users.

In general, WiFi performs better than GPS sampled every 60 seconds (GPS 60) and worse than GPS 30 (it is approximately equivalent to GPS 40). Hence, if, on a hypothetical device, GPS 30 uses less energy than WiFi, GPS 30 should always be used. Our numbers above suggest that for an iPhone with the ability to duty cycle like the Bluetooth GPS (i.e., taking 6 seconds to acquire a fix), GPS 30 would use only about 1/5th of the power of continuous GPS sampling, which would still make it about 5× more expensive than WiFi sampling. However, it would extend the life of the phone to about 5 hours with the screen on (much better than the 2.5 hours with continuous GPS sampling). Since the delay prediction errors of GPS 30 are substantially better—with shortest path delay estimate errors of less than 10% for GPS 30 and almost 20% for WiFi at the 90th percentile—this suggests that GPS 30 might be a better strategy in such cases.

Similarly, Figure 11 shows that for hotspot detection, if WiFi sampling is very cheap, like on the iPhone (or "free" because WiFi is in use anyway) it may be a good idea to use GPS 30 + WiFi (a 5× reduction in power from GPS) rather than GPS 20 (only a 3.3× reduction).

Offline Energy Optimization. Given a characterization of the sampling costs of WiFi and GPS on any device and a power budget, our results make it possible to perform this kind of analysis to derive an optimal sampling strategy for that device. For any device we want to run on, we measure the energy costs of continuously sampling GPS (g) and WiFi (w) and determine the ratio g/w of the power consumed by a GPS fix to that consumed by a WiFi scan. Now, given a target power budget p and the ratio g/w, we perform an analysis to determine the best sensor(s) and sampling strategy to use on that device to maximize accuracy for our target application (routing or hotspot detection) for that combination of p and g/w. The options we consider are GPS every k seconds for some k, WiFi or a combination of GPS every k seconds with WiFi. The following table shows the results of solving this optimization problem for some sample values of p and g, assuming the WiFi cost w is pegged to 1 unit:

GPS Cost	Power Budget	Optimal Strategy
0.5	0.1	GPS 30
6	1	GPS 36
7	1	WiFi
24.9 (iPhone)	5	GPS 30
24.9 (iPhone)	3.5	GPS 60 + WiFi

For example, a power budget of $p = 2$ units in the table means that we want to use at most twice the power consumed by continuous WiFi sampling. Clearly, some settings of these parameters don't make sense – for example, $p < 1$ and $g > 1$ – so we only consider meaningful parameter settings. Results are presented for the route planning application (in the table, *GPS k* means sampling GPS every k seconds and *GPS k + WiFi* means sampling GPS every k seconds, and WiFi all the time). We see that there is a threshold of GPS cost g beyond which using WiFi is the best option (as one would expect). Also, given a small power budget on a phone like the iPhone where GPS is power-hungry, the optimal strategy starts to use WiFi.

Figure 13 illustrates the solution to the optimization problem as a function of the power budget $\frac{p}{w}$ and the GPS sampling cost $\frac{g}{w}$, both expressed as ratios to WiFi sampling cost. First look at the case when $p = w$, i.e., the part of the graph along the x axis. Here, it is impossible to use both GPS and WiFi together, and there is a clear choice between *GPS k* and WiFi. For values of g below a certain threshold, *GPS k* is preferable and for values above, WiFi is preferable. Next, when $p \geq w$, $GPS + WiFi$ is always preferable to just WiFi, because the additional GPS points that can be sampled with the extra power budget never hurt accuracy.

The next choice we consider is between *GPS k* for some sampling interval k, and $GPS\ k' + WiFi$ at a higher sampling interval k', where k' is chosen so that the energy costs of the approaches are equal. First consider the case when $\frac{g}{w}$ is large—then the cost of WiFi is negligible, and for any k, *GPS k* and $GPS\ k + WiFi$ have approximately the same energy consumption, so it purely boils down to which one is better in terms of accuracy. In our empirical data, for approximately $k = 20$ ($k_{critical}$ in the graph), using interpolated GPS is preferable to using both GPS and WiFi together. Hence, whenever the power budget exceeds approximately that of GPS 20 (the dotted line) it is preferable to use only subsampled GPS. The graph also shows that for lower values of g, or beyond a certain power budget, it is better to use the power to increase the GPS subsampling rate (i.e., to reduce k) than to use WiFi.

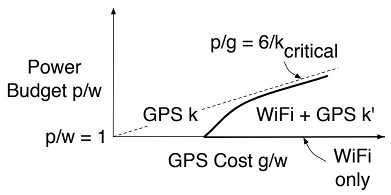


Figure 13: Optimal strategy for different energy cost regimes.

Of course, our accuracy results for routing and hotspot accuracy, and hence the optimal strategy, also depend on other factors. In a non-urban setting with lower road density, lower GPS sampling rates may be sufficient to estimate these parameters. There may also be fewer WiFi hotspots, which would make WiFi a less viable strategy as it would be even more prone to long outages. At the opposite extreme, in very dense urban environments, GPS may perform worse than indicated by our results, as "urban canyon" effects from tall buildings impair its accuracy.

5.5 Impact of Noise

In this section, we look at the performance of our map-matching algorithm for varying levels of noise in the input data. We compare our algorithm to the simple approach of simply matching each point to the nearest segment. We demonstrate that our algorithm is relatively robust to noise, much more so than nearest-segment matching. We believe this is the first evaluation of map matching in the presence of noise.

In these experiments, we again use the clean drives described in Section 5.1. Each point in each drive is labeled with a ground truth segment. We generate a *perturbed* version of each cleaned drive by adding random zero-mean Gaussian noise with variances of 15, 40, and 70 meters to both the X and Y coordinates of each point in the drive (40 meters roughly corresponds to WiFi). We measure the point error rate (PER) of our Viterbi-based HMM and the simple nearest-segment match on each of the perturbed drives. PER is the fraction of points on a drive assigned to the wrong segment. A CDF of PER over all the perturbed drives is shown in Figure 14.

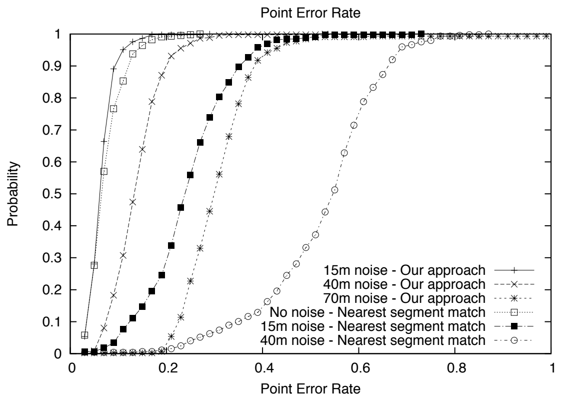


Figure 14: HMM vs Nearest Segment Matching.

The results show that our approach performs very well with 15 meter noise—about as well as nearest-neighbor matching with no noise. The median PER is less than 5%, and the 90th percentile PER is less than 8%. For 40 meter noise (corresponding to WiFi localization), these values are about 8% and 10%. Nearest segment matching performs much worse—with just 15 meters noise, the median point error rate is almost 20%. At 70 meters noise, our algorithm does better than nearest neighbor with 40 meters noise, achieving a median PER of about 20%. In general, the HMM does better than nearest neighbor matching for the reasons described in Section 3; in particular, in cases like that in Figure 5, the HMM is able to correctly assign noisy point P2 (which is nearest to S2) to S3.

Beyond 70 meters noise (not shown), our algorithm performs very poorly, making many errors. This suggests that HMM-based algorithms and applications like route planning and hotspot detection are poorly suited to map matching when using location estimates from cellular triangulation, which has errors in excess of 100 meters, at least when performing predictions in dense urban road networks. It is possible that the HMM with such noisy data might perform better in less dense networks (e.g., just freeways or rural areas.)

6 Related Work

The Mobile Millennium project at UC Berkeley [1] has built software to report traffic delays on mobile phones, as well as to use mobile phones as probes for detecting traffic. They focus on real-time traffic reporting. In [6] and [7] the authors develop a model for predicting flows and delays in the near future from traces of probe data. However, they assume GPS data and do not look into the effects of noisy data.

The NeriCell project [20] focuses on monitoring road conditions and traffic using smartphones. Their goal is to combine data from GSM localization, GPS, and accelerometers to get a picture of road surface quality as well as traffic conditions, such as locations where users are braking aggressively. Their primary contributions are in processing accelerometer data, as well as power management. They do not provide algorithms for travel time estimation or map matching.

Using HMMs for map matching has been proposed in [16] and [19]. However, their research focus only on using frequently-sampled GPS data with low noise. To the best of our knowledge, there have been no quantitative studies of their accuracy.

Gaonkar et al. [9] present the idea of a "micro-blog" where users can annotate locations and pick up other users' annotations as they travel. To implement this, they use an energy-efficient continuous location sampling technique, but focus on providing approximate estimates of location, rather than performing map matching or estimating travel time.

Yoon et al. [25] use GPS data to classify road conditions as "good" or "bad," both spatially and temporally (which reflect the steadiness and speed of traffic, respectively). They take frequently-sampled GPS points and calculate how a car's delay is distributed over each segment. This "cumulative time-location" data is converted to spatio-temporal data and then classified. This work differs from ours in that it assumes the ability to get relatively accurate travel time estimates on individual segments. Their method would likely fail on noisy data such as WiFi, because the cumulative time-location measurements would be incorrect.

Using approaches inspired by the notion of *k-anonymity* [23], Gruteser and Grunwald [10] show how to protect locational privacy using spatial and temporal cloaking. A number of recent works show how to protect locational privacy while collecting vehicular traffic data [13, 18, 11] and in GPS traces [14]. In addition, some recent papers [12, 14] have developed tools to quantify the degree of mixing of cars on a road needed to assure anonymity (notably the "time to confusion" metric). The virtual triplines scheme [12] proposes a way to determine when it is "safe" from the standpoint of privacy for a vehicle to report its position using such a quantification. Many of these techniques could be used in VTrack.

7 Conclusions

This paper presents VTrack, a system for using mobile phones to accurately estimate road travel times using a sequence of inaccurate position samples, and evaluates it on route planning and hotspot detection. Our approach addresses two key challenges: 1) reducing energy consumption using inaccurate position sensors (WiFi rather than GPS), and 2) obtaining accurate travel time estimates from these inaccurate positions. VTrack uses an HMM-based map matching scheme, with a way to interpolate sparse data to identify the most probable road segments driven by the user and to attribute travel times to those segments. We presented a series of results that showed our approach can tolerate significant noise and outages in these estimates, and still successfully identify highly delayed segments, and provide accurate enough travel time estimates for accurate route planning.

There are a couple of interesting directions for future work. We plan to develop an online, adaptive algorithm that dynamically selects the best sensor to sample taking available energy and the current uncertainty of the node's position and trajectory into account. Second, improving the quality of today's algorithms to predict future travel times on segments, using historical travel times and sparse amounts of real-time data, would be useful for traffic-aware routing.

8 Acknowledgments

This work was funded by the National Science Foundation under grants CNS-0720079, CNS-0520032, NSF IIS-0448124 and CNS-0509261.

References

[1] The mobile millenium project. http://traffic.berkeley.edu.

[2] Navteq. http://navteq.com/about/data.html.

[3] Bureau of Transportation Statistics. http://www.bts.gov.

[4] The CarTel project. http://cartel.csail.mit.edu/.

[5] Y. chung Cheng, Y. Chawathe, A. Lamarca, and J. Krumm. Accuracy characterization for metropolitan-scale wi-fi localization. In *In Proceedings of Mobisys 2005*, pages 233–245, 2005.

[6] C. Claudel and A. Bayen. Guaranteed bounds for traffic flow parameters estimation using mixed lagrangian-eulerian sensing. In *Allerton Conference on Communication, Control, and Computing*, 2008.

[7] C. Claudel, A. Hofleitner, N. Mignerey, and A. Bayen. Guaranteed bounds on highway travel times using probe and fixed data. In *88th TRB Annual Meeting Compendium of Papers*, 2009.

[8] D. F.V. Gps accuracy: Lies, damn lies, and statistics. 1998.

[9] 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 *MobiSys*, pages 174–186, 2008.

[10] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *ACM MobiSys*, 2003.

[11] M. Gruteser and B. Hoh. On the anonymity of periodic location samples. In *Pervasive*, 2005.

[12] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, 2008.

[13] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing security and privacy in trafc-monitoring systems. *IEEE Pervasive Computing*, 5(4):38–46, 2006.

[14] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Preserving privacy in GPS traces via uncertainty-aware path cloaking. In *ACM CCS*, 2007.

[15] 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.

[16] B. Hummel. Map matching for vehicle guidance. In *Dynamic and Mobile GIS: Investigating Space and Time*. CRC Press: Florida, 2006.

[17] Inrix home page. http://www.inrix.com/.

[18] J. Krumm. Inference attacks on location tracks. In *Pervasive*, 2007.

[19] J. Krumm, J. Letchner, and E. Horvitz. Map matching with travel time constraints. In *SAE World Congress*, 2007.

[20] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008.

[21] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West. PEIR, The Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research, 2009.

[22] D. Sperling and D. Gordon. *Two Billion Cars: Driving Toward Sustainability*. Oxford University Press, 2009.

[23] L. Sweeney. k-anonymity: A model for protecting privacy, 2002.

[24] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*, 1967.

[25] J. Yoon, B. Noble, and M. Liu. Surface Street Traffic Estimation. In *MobiSys*, 2007.